Ein neues Paradigma für Big Data

Sophia Schlöter
Katharina Littau
Tom Holöchter
David Biskup
Jost Fröbrich
Till Fritsche
Tilko Bohms
Tobias Nünning

Agenda

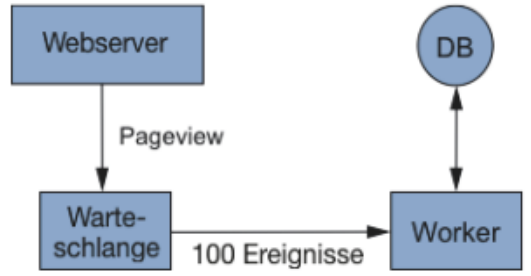
- **Skalierung einer herkömmlichen Datenbank**
- NoSQL
- Eigenschaften eines Big Data-Systems
- Inkrementelle Architekturen
- Lamdba-Architektur
- Trendentwicklungen

1.2 Skalierung mit einer herkömmlichen Datenbank

- Herkömmliche Datenbanktechnologie stößt an ihre Grenzen
  - Skalierbarkeit
  - Komplexität

- Skalierung mit einer Warteschlange
  - Hohe Arbeitslast der DB durch einzelne Requests
  - Stapelverarbeitung
    - Warteschlange zwischen Webserver und DB
    - Requests werden zusammengefasst
    - Worker-Prozess: Zusammengefasste Datenbankaktualisierung

Spaltenname	Typ
id	integer
user_id	integer
url	varchar(255)
pageviews	bigint



1.2 Skalierung mit einer herkömmlichen Datenbank

- Skalierung durch Sharding
  - Verteilung der Datenbanktabellen mehrerer Datenserver
  - Gleichmäßige Verteilung durch Zuordnung einer Hashfunktion
  - Library als Schnittstelle zum Datenbankcode
  - Zusammenführung der Ergebnisse der verteilten Datenbanken
- Erste Probleme mit der Fehlertoleranz
  - Festplattenausfall der Server durch hohe Anzahl an DB
  - Separate Warteschlangen für nicht verfügbare DB
  - Durchführung von Replikationen und Hinzufügen von Slave-DB
- Probleme fehlerhafter Daten
  - Bugs und schwierige Identifikation fehlerhafter Datensätze

1.2 Skalierung mit einer herkömmlichen Datenbank

- Was ist schief gegangen
  - Zunehmende Komplexität der Anwendung
    - Warteschlangen, zusätzliche DB, Skripte zur Umverteilung, …
  - Keine Verhinderung menschlichen Versagens/Bugs

- Inwiefern sind Big-Data-Verfahren hilfreich
  - DB wissen, dass sie Komponente eines verteilten Systems sind
  - Netzknoten für Skalierung – automatische Umverteilung
  - Konzept der unveränderlichen Daten

NoSQL ist kein Wundermittel

- NoSQL Datenbanken wie Cassandra oder Riak glänzen vor allem bei enorm großen Datenmengen und lässt sich besser skalieren.
- Diese NoSQL Datenbanken haben jedoch Probleme bei der Strukturierung der Daten.
- Neuen Anwendungen in die Datenbank einzupflegen wird dadurch schwieriger.
- Jedoch ist bei intelligent miteinander verknüpfen, können skalierbare Systeme minimaler Komplexität für beliebige Inhalte erstellen, die bei menschlichem Versagen fehlertolerant reagieren.

Erwünschte Eigenschaften eines Big-Data Systems

- **Belastbarkeit & Fehlertoleranz**
  - Schwierigkeiten: ausfallende Maschinen, Datendoppelungen, parallele Ausführung etc.
  - Lösung: Möglichst die Probleme umgehen

  = unveränderliche Daten und Neuberechnungen von Werten (=keine Fehler vom Menschen)
  - Big-Data-System wird übersichtlicher und einfacher bei der Datenwiederherstellung

- **Lesen & Aktualisieren mit geringen Latenzzeiten**
  - Um System nicht zu überlasten: Latenzzeiten möglichst gering halten im Big-Data-System

  = Je nach Fall unterschiedliche Latenzzeit – sollte einberechnet werden

- **Skalierbarkeit**

  = Geschiwindigkeit bei wachsenden Datenmengen und steigender Arbeitslast aufrechterhalten
  - Lambda-Architektur: drei Layer horizontal skalierbar

- **Allgemeingültigkeit**
  - Lambda-Architektur: Funktionen, die von der Gesamtheit der Daten abhängen

  = Somit einsetzbar in vielen Bereichen: z.B. Finanzverwaltung, Analyse soziale Netzwerke etc.

Erwünschte Eigenschaften eines Big-Data Systems

- **Erweiterbarkeit**
  - Big-Data Systeme sollten leicht erweiterbar sein
  - Neue Funktionen sollten unter geringem Aufwand implementierbar sein
- **Ad-Hoc Abfrage**
  - Beliebige Abfragen sollten unkompliziert gestartet werden können
- **Minimaler Wartungsaufwand**
  - Wartung ist Last, bei Erweiterung von Maschinen muss alles ordnungsgemäß arbeiten
  - Um Aufwand zu minimieren sollten unkomplizierte Komponenten gewählt werden
  - Man muss die Arbeitsweise des Systems verstehen um Fehler zu beheben und Einstellungen vornehmen zu können
- **Fehlerbehebung**
  - Es müssen Informationen zur Fehlerbehebung gegeben sein
  - Wie jeder Wert zustandegekommen ist, muss nachverfolgbar sein

1.6 Schwierigkeiten vollständig inkrementeller Architektur

- Zum besseren Verständnis zu inkrementell: Bei einem inkrementellen Backup werden nur die Daten gesichert, die sich seit dem letzten Backup – egal welchen Typs – verändert haben. Meistens wird dazu die Zeitangabe zur letzten Änderung einer Datei mit dem Datum des letzten Backups verglichen. Vorteil: Weniger Datenplatz wird benötigt, da nur "neue" Daten abgespeichert werden müssen und kein vollständiges BackUp erstellt werden muss.

- 1.6.1 Komplexität im Betrieb

- bei les- und beschreibbaren Datenbank wird der zugehörige Index beim Hinzufügen und Ändern von Datensätzen ständig modifiziert --> dadurch werden alte Teile nicht mehr benötigt, verwenden aber nach wie vor Speicherplatz --> Speicherplatz muss früher oder später wieder freigegeben werden, wenn Festplatte nicht vollaufen soll--> Lösung: Komprimierung

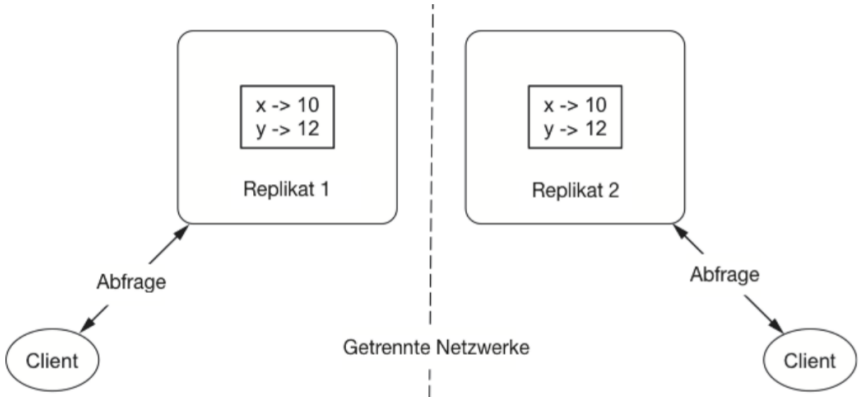
- Komprimierung ist aufwendiger Prozess (hohe Arbeitslast für CPU und Festplatte) --> dadurch sinkt Geschwindigkeit während Komprimierungsvorgang --> wenn zu viele Maschinen gleichzeitig Komprimierung vornehmen, kann Server überlasten --> um das zu verhindern, muss geplant werden, wann welcher Netzknoten eine Komprimierung vornimmt

1.6 Schwierigkeiten vollständig inkrementeller Architektur

- 1.6.2 Extreme Komplexität, um letztendliche Konsistenz zu erzielen

- weitere Komplexität inkrementeller Datenbanken wird sichtbar, wenn man Hochverfügbarkeit benötigt

--> Hochverfügbarkeit konkurriert mit Konsistenz (Konsistentes System liefert immer sämtliche vorangegangenen Schreibaktivitäten) --> es ist unmöglich, sowohl Hochverfügbarkeit als auch Konsistenz zu gewährleisten --> dadurch liefert ein hochverfügbares System teils veraltete Ergebnisse



--> um das Problem mit den veralteten Ergebnissen bei hochverfügbaren Systemen zu lösen, muss Wert + Zeitpunkt (seit wann Werte voneinander abweichen) übertragen werden --> zusätzlich muss Code geschrieben werden, der Werte anschließend aus diesen Informationen korrigiert

1.6. Keine Fehlertoleranz gegenüber menschlichem Versagen

- Inkrementelles System speichert den Zustand in der Datenbank dauerhaft
- Datenbank mit fehlerhaften Dateien, wenn Menschen Fehler machen
- Problemlösung: asynchrone Architektur
- Neuerungen gehen zunächst in eine Warteschlange
- Bei Fehlern kann die vorherige Datenbank rekonstruiert werden

1.6. Vollständig inkrementelle Lösung kontra Lambda-Architektur

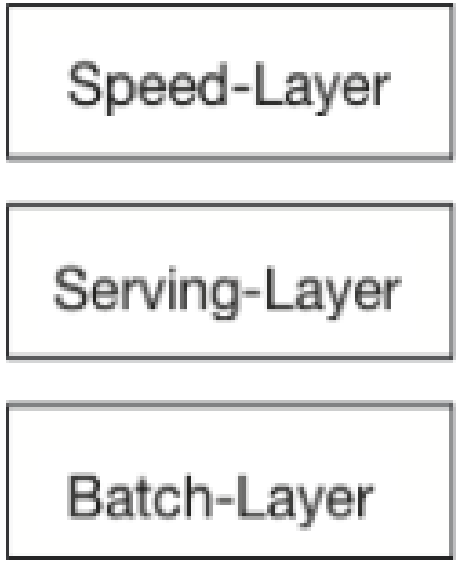
- Praxisproblem: Berechungen von Pageviews
- 2 Arten von eingehenden Daten

1. Pageviews: eine User-ID, eine URL und einen Zeitstempler
2. Equivs: zwei User-IDs, allerdings von der selben Person

- Ziel ist es die Anzahl der unterschiedlichen Personen zu tracken
- Inkrementelle Lösung oder Lamba-Architektur-Lösung
- Die inkrementelle Lösung ist deutlich fehleranfälliger und zeitaufwendiger

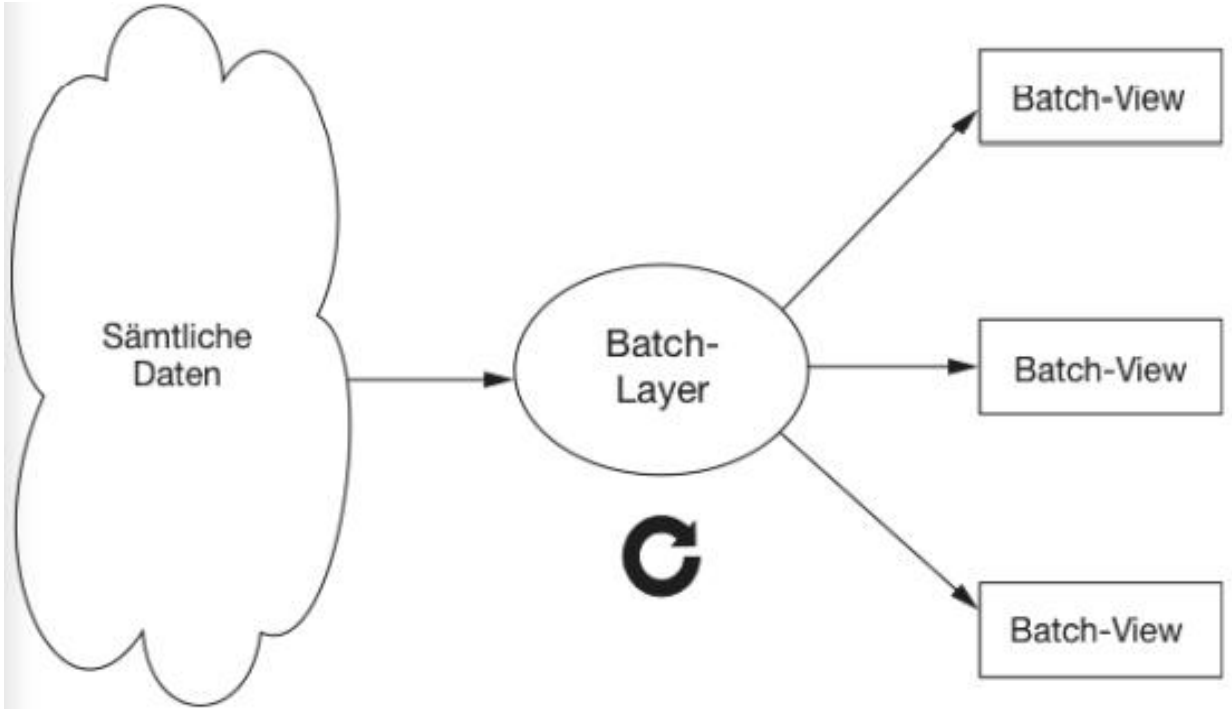
1.7 Lambda-Architektur

- Herausforderung: beliebige Funktionen mit beliebigen Eingabedaten in Echtzeit berechnen
- Idee der Lambda-Architektur: Big-Data-Systeme werden in diversen Layer eingeteilt
  - Jeder Layer übernimmt eine bestimmte Teilmenge der Funktionen
  - Bestmögliche Performance und Belastbarkeit
- Alternative: Ausführung einer Funktion im laufenden Betrieb
  - Enorme Ressourcen
  - Unangemessener Aufwand



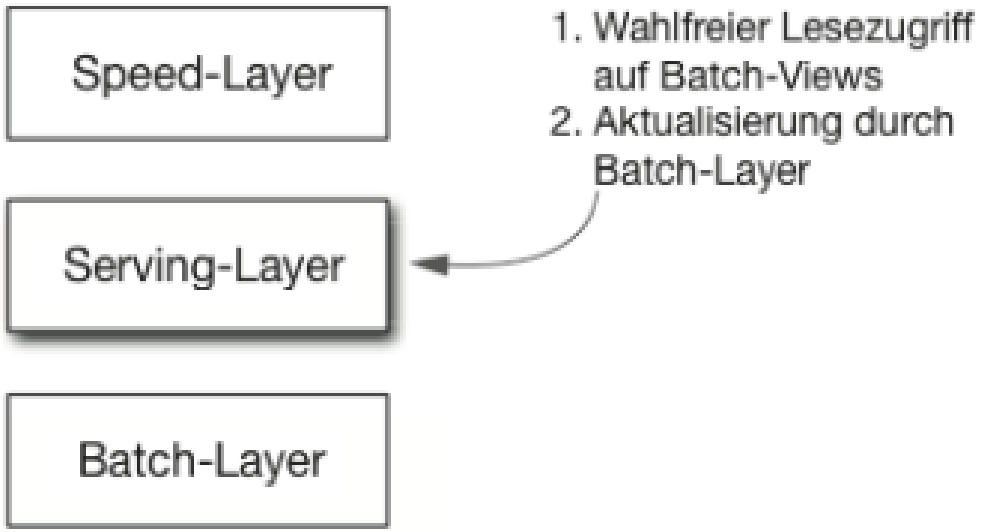
1.7 Batch-Layer

- Speichert den Stammdatensatz
- Vorabberechnung der Abfragefunktionen mit gespeicherten Daten (Ergebnisse werden als Batch Views gespeichert)
  - ➢ Daten können während der Berechnung bereits veraltet sein
- Läuft in einer while(true)-Schleife und berechnet die Batch-Views immer wieder neu



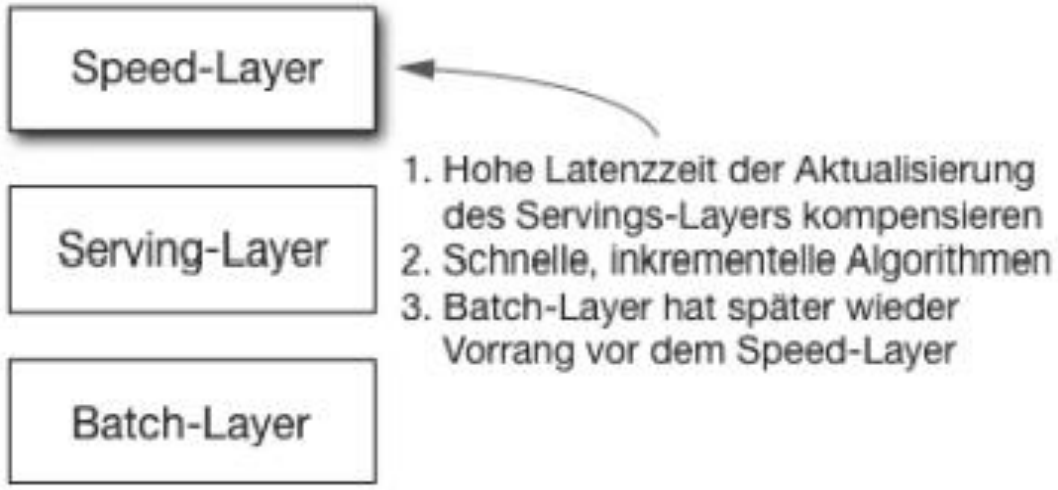
1.7 Serving-Layer

- Datenbank, die Batch-Views lädt und Lesezugriff darauf ermöglicht
- Schreibzugriff wird nicht unterstützt, daher ist die Komplexität gering
- Sollten neue Batch-Views bestehen, werden diese automatisch aktualisiert



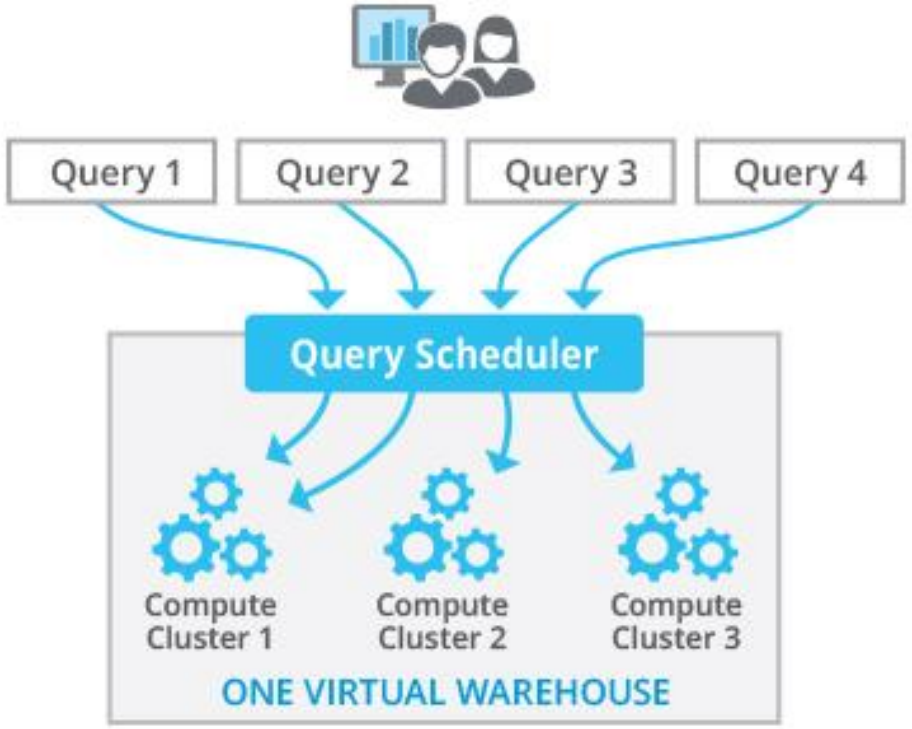
1.7 Speed-Layer

- Batch-Views enthalten nicht die Daten, die während einer Berechnung eintrafen
- Diese Lücke wird durch den Speed-Layer geschlossen
- Ähnelt dem Batch-Layer insofern, als dass er anhand eingehender Daten Views erzeugt
- Unterschied: Der Speed-Layer berücksichtigt nur die letzten Daten, während der Batch-Layer auf sämtliche Daten zurückgreift
- Um möglichst geringe Latenzzeiten zu erzielen, aktualisiert der Speed-Layer immer nur die Echtzeit-Views beim Eingang neuer Daten, statt die Daten von Grund auf neu zu berechnen (inkrementelle Berechnungen)



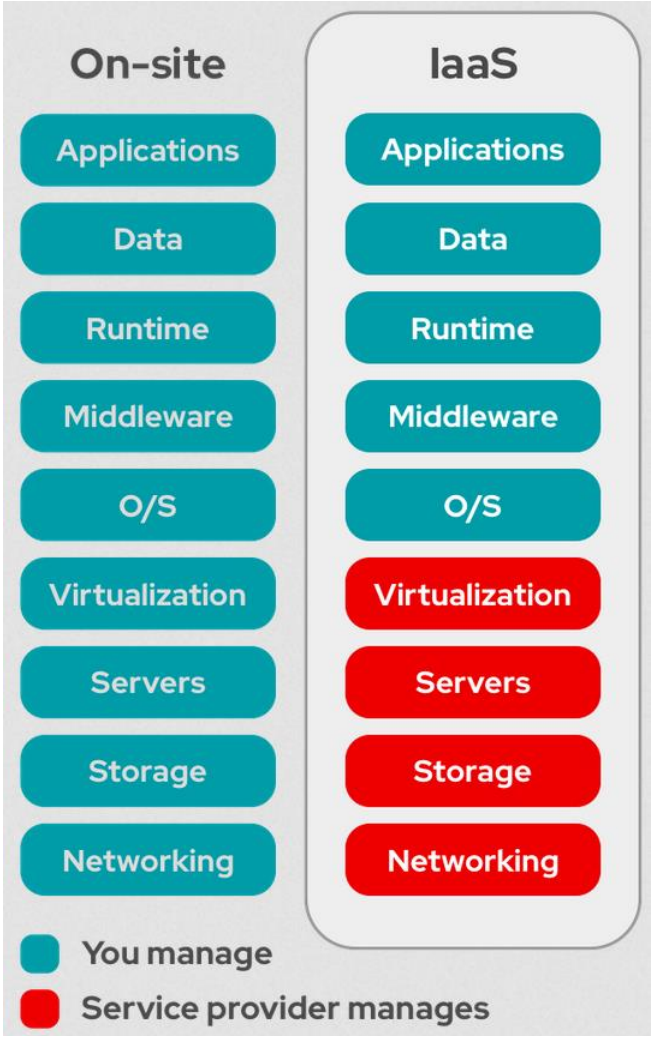
1.8 Die neusten Trends - Prozessoren

- Physikalische Grenze bei Prozessorkernen erreicht
  - Schnellste CPU 2021 ist ein AMD Ryzen™ Threadripper™ 3990X (64 Kerne)
  - Ein Prozessorkern führt eine Aufgabe aus
- Komplexe Berechnungen müssen verteilt werden (Multi-Threading, Clustering)
- Neue Frameworks für die optimierte Berechnung großer Datenmengen
- Erweiterung der Rechenleistung nicht mehr horizontal sondern vertikal



1.8 Die neusten Trends - Elastic Clouds

- Infrastructure as a Service (IaaS)
  - Keine eigene Hardware
  - Abtretung der Verantwortung
  - Maximale Skalierbarkeit ("on-the-fly")
  - Keine Bindung
- Populäres Beispiel AWS (Amazon Web Services)
  - Diverse Anwendungen, Datenbankdienste und vorkonfigurierte Big Data Lösungen
  - Minimierung der Wartung
  - "Spotinstances" bieten kurzfristig günstige Rechenleistung



1.8 Die neusten Trends – Open Source

Große Open Source Community mit der Entstehung und Verbesserung vieler wichtiger Anwendungen

Stapelverarbeitungssysteme

Serialisierungsframework

NoSQL-Datenbanken

Echtzeitberechnungssysteme