

## Verarbeitung großer Datenmengen

Motivation

MapReduce

Zusammenfassung



- Problemstellungen (Beispiele)
  - Indexierung des World Wide Web
  - PageRank-Berechnungen für Web-Seiten
  - Erstellung von Log-Statistiken
  - ...
- Mögliche Herangehensweisen
  - Trennung zwischen Speicherung von Daten und ihrer Verarbeitung
    - Vergleiche: Windows Azure Storage und Windows Azure
    - Separate Skalierbarkeit der Subsysteme möglich
    - Erhöhter Aufwand für Datenübertragung erforderlich
  - Speicherung und Verarbeitung von Daten erfolgt auf denselben Rechnern
    - Vergleiche: Google File System und MapReduce
    - Ausnutzen von Datenlokalität
    - Anwendung muss Einblick in Interna des Datenspeichersystems besitzen




## Herausforderungen

- Ausgangssituation
  - Sehr große Datenmengen
  - Hohe Anzahl an Rechenknoten
- Ziele
  - Ausnutzung der zur Verfügung stehenden Kapazitäten
  - Einfache Realisierung von Anwendungen
- Herausforderungen
  - Wie lässt sich ein System aufbauen, das es ermöglicht, mit vergleichsweise wenigen Code-Zeilen große Datenmengen zu verarbeiten?
  - Wie erspart man einem Anwendungsprogrammierer sich um Aspekte wie Verteilung, Parallelisierung und Fehlertoleranz kümmern zu müssen?
  - Wie lässt sich Wissen über das zugrundeliegende Datenspeichersystem zur Entlastung von Netzwerkverbindungen nutzen?



## MapReduce

- Konzept
  - Kombination aus Programmiermodell und Framework
  - Grundidee
    - Framework übernimmt Verteilung der Anwendung
    - Programmierer implementiert zwei Methoden
      - \* Map: Abbildung der Eingabedaten auf Schlüssel-Wert-Paare
      - \* Reduce: Zusammenführung der von Map erzeugten Schlüssel-Wert-Paare
- Implementierungen
  - Google MapReduce (nicht öffentlich verfügbar)
  - Apache Hadoop MapReduce (→ siehe Übung)
  - Phoenix (Stanford University)
  - ...
- Literatur
  -  Jeffrey Dean and Sanjay Ghemawat  
**MapReduce: Simplified data processing on large clusters**  
*Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04), S. 137–150, 2004.*



# Framework

- Rahmenbedingungen bei Google
  - Cluster aus Hunderten bzw. Tausenden von Commodity-Rechnern
  - Datenvolumen übersteigt den Platz im Hauptspeicher bei weitem
  - Datenverwaltung mittels Google File System
- Aufgaben des MapReduce-Framework
  - Partitionierung der Eingabedaten
  - Scheduling von Anwendungsprozessen
  - Fehlerbehandlung bei Ausfällen
  - Kommunikation zwischen Rechnern
- Bereitstellung als Bibliothek
- Literatur
  - 📄 Luiz André Barroso, Jeffrey Dean, and Urs Hölzle  
**Web search for a planet: The Google cluster architecture**  
*IEEE Micro*, 23(2):22–28, 2003.



# Begrifflichkeiten nach [Dean et al.]

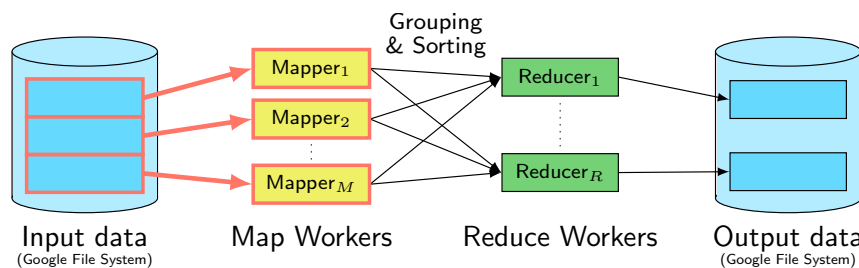
- Rechner im Cluster: Worker-Rechner (*Worker Machine*)
- MapReduce-Job
  - Vom Nutzer an das Framework übermittelte Aufgabe
  - Aufspaltung in Teilaufgaben (*Tasks*)
    - *Map-Task*: Aufgabe, einen Teil der Eingabedaten zu verarbeiten
    - *Reduce-Task*: Aufgabe, einen Teil der Zwischenergebnisse zusammenzufassen
- Framework-Prozesse auf Worker-Rechnern
  - *Master*-Prozess
    - Dedizierter Prozess zur Verwaltung des Framework
    - Aufgabe: Zuweisung von Map- und Reduce-Tasks zu Worker-Prozessen
  - *Worker*-Prozesse
    - Restliche Prozesse
    - Aufgabe: Ausführung von Map- und Reduce-Tasks
    - Benennung je nach übernommener Aufgabe: *Map*- bzw. *Reduce-Worker*



# Verarbeitungsablauf

## Vorbereitung

1. Nutzer übermittelt Job an einen Job-Scheduler
2. Scheduler: Auswahl von Worker-Rechnern zur Bearbeitung des Jobs
3. MapReduce-Bibliothek
  - Annahme:  $M$  Map-Tasks,  $R$  Reduce-Tasks
  - Partitionierung der Eingabedaten in  $M$  etwa gleichgroße Teile (16-64 MB)
  - Verteilung des Programms auf Worker-Rechner
  - Start des Master-Prozesses bzw. der Worker-Prozesse
4. Master: Zuteilung von Map- und Reduce-Tasks zu Worker-Prozessen



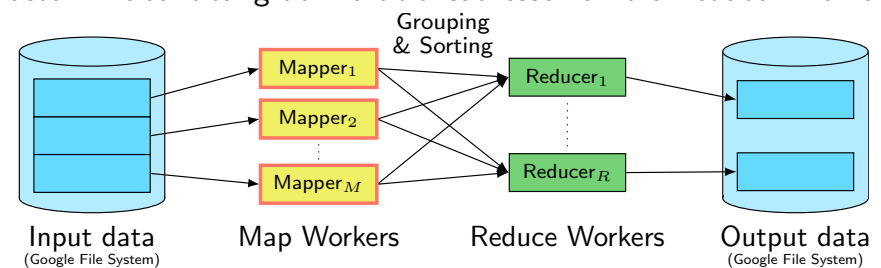
# Verarbeitungsablauf

## Map-Phase

5. Map-Worker
  - Einlesen der Eingabedatenpartition aus dem Google File System
  - Konvertierung der Eingabedaten in Schlüssel-Wert-Paare
  - Aufruf der Map-Funktion für jedes der Schlüssel-Wert-Paare

`map(KeyM, ValueM) → List<{KeyR, ValueR>`

  - Puffern der Zwischenergebnisse im Hauptspeicher
  - Periodisches Schreiben der Zwischenergebnisse auf die lokale Festplatte
    - Aufteilung in  $R$  Partitionen mittels *Partitionierungsfunktion* [z. B.  $\text{hash}(\text{Key}_R) \% R$ ]
    - Meldung der Partitionsadressen an den Master
6. Master: Weiterleitung der Partitionsadressen an die Reducer-Worker



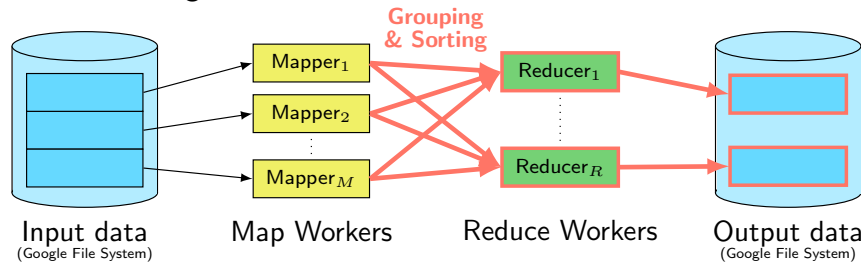
## 7. Reduce-Worker

- Holen der Zwischenergebnisse per Fernaufruf
- Sobald alle benötigten Zwischenergebnisse lokal vorhanden sind
  - Sortierung der Zwischenergebnisse nach Schlüssel
  - Gruppierung aller zum selben Schlüssel gehörigen Werte
- Aufruf der Reduce-Funktion für jede Schlüssel-Werte-Gruppe

`reduce(KeyR, List<ValueR>) → List<ValueR>`

- Sicherung der Ausgabedaten im Google File System
- Eine Ausgabedatei pro Reduce-Task (→ keine Zusammenführung)

## 8. Master: Meldung an Nutzer sobald alle Tasks beendet wurden

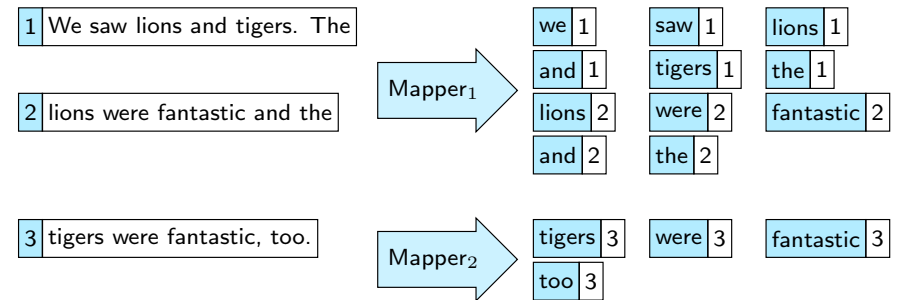


## ■ Anwendungsbeispiele

- Wörter zählen (→ siehe Übung)
- Verteiltes grep
- Verteiltes Sortieren
- Invertierter Index

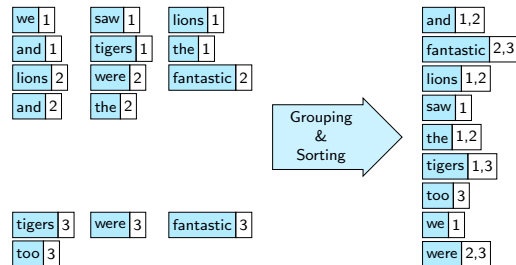
## ■ Beispiel: Zeilenindex des ersten Auftretens eines Wortes [case-insensitive]

### Map-Phase (2 Map-Worker, jeweils 1 Map-Task)

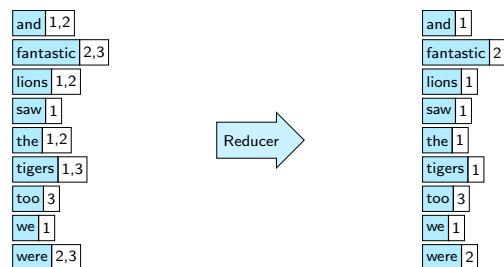


# Zeilenindex des ersten Auftretens eines Wortes

## Gruppierung und Sortierung



## Reduce-Phase (1 Reduce-Worker, 1 Reduce-Task)



# Implementierung

## ■ Anpassung des Framework durch den Nutzer

- Verwendung nutzerdefinierter Datentypen
  - Abbildung der Eingabedaten auf Schlüssel-Wert-Paare
  - Einfluss auf Format der Ausgabedaten möglich
- Nutzerdefinierte Partitionierung der Zwischenergebnisse
  - Standard: Aufteilung anhand eines Hash-Werts über den Schlüssel
  - Bereitstellung einer eigenen Abbildungsvorschrift
  - Einfluss auf Zuordnung der Ergebnisse zu Ausgabedateien möglich

## ■ Einführung nutzerdefinierter Zähler

- Einsatz bei statistischen Auswertungen
- Zugriff auf Zähler in der Map- und/oder Reduce-Funktion
- Zusammenfassung der Zähler einzelner Tasks im Master

## ■ Bereitstellung von Statusinformationen

- Master-Prozess verfügt über eigenen HTTP-Server
- Übersicht über aktuellen Job-Fortschritt (z. B. Anzahl beendeter Tasks)

## Fehlertoleranz

- Master: Maßnahmen zur Tolerierung von Worker-Ausfällen
  - Periodische Ping-Anfragen an Worker-Prozesse
  - Falls Worker  $w$  nicht antwortet →  $w$  wird als „ausgefallen“ definiert
    - Alle  $w$  zugeteilten Map-Tasks werden an andere Worker-Prozesse vergeben
    - Alle  $w$  zugeteilten Reduce-Tasks, die dem Master noch nicht als beendet gemeldet wurden, werden an andere Worker-Prozesse vergeben
    - Reduce-Worker werden über die Neuzuteilung benachrichtigt
  - Prinzip: Einfachheit vor Effizienz
    - Kein Versuch eventuell bereits vorhandene Zwischenergebnisse zu retten
    - Redundante Bearbeitung von Tasks wird in Kauf genommen
- Ausfall des Master-Prozesses
  - Keine Fehlertoleranzmechanismen
    - Ausfall wird in Kauf genommen
    - Annahme: Nutzer startet seinen MapReduce-Job neu
  - Mögliches Vorgehen: Periodische Sicherungspunkte des Master-Zustands



## Optimierungen

- Datenlokalität
  - Ziel: Einsparung der übers Netzwerk zu sendenden Daten
  - Ansatz: MapReduce-Master berücksichtigt Speicherort der Eingabedaten bei der Zuteilung von Tasks zu Worker-Prozessen
  - Beispiel: Ausführung eines Map-Task auf einem Worker-Rechner, auf dem das Google File System ein Replikat der Eingabepartition verwaltet
- Task-Granularität
  - Ziel: Verbesserte Lastbalancierung, beschleunigte Fehlerbehandlung
  - Ansatz: Aufspaltung eines Jobs in viele Tasks
  - Vorteile
    - Feingranulare, dynamische Task-Platzierung nach Lastkriterien möglich
    - Bei Ausfall: Verteilung der Tasks auf viele Worker-Prozesse
  - Nachteile
    - Zusätzliche Scheduling-Entscheidungen für den Master-Prozess
    - Ungünstiger Ansatz für Reduce-Tasks → große Anzahl an Ausgabedateien
  - Beispiel [Dean et al.]: 200.000 Map-, 5.000 Reduce-Tasks (2.000 Rechner)




## Optimierungen

- Redundante Task-Ausführung
  - Problem
    - In der Praxis benötigen einige wenige Worker-Prozesse deutlich länger als alle anderen für die Bearbeitung ihrer Tasks → „Nachzügler“ (*Stragglers*)
    - Mögliche Gründe: Überlast auf dem Rechner, Hardware-Fehler,...
  - Verzögerungen bei der Bearbeitung des MapReduce-Jobs
  - Lösung
    - Sobald ein Großteil aller Tasks beendet ist, vergibt der Master die sich noch in Ausführung befindenden Tasks an weitere Worker-Prozesse → *Backup-Tasks*
    - Verwendung der Ergebnisse des (Original-/Backup-)Task, der zuerst fertig ist
- Zusammenfassen von Zwischenergebnissen
  - Ziel: Reduktion der Zwischenergebnisse → Entlastung des Netzwerks
  - Ansatz: Spezifizierung einer *Combiner*-Funktion
    - Vorverarbeitung der Zwischenergebnisse während der Map-Phase
    - Meist identisch mit der Reduce-Funktion



## Zusammenfassung

- MapReduce
  - Programmiermodell
    - Map: Abbildung der Eingabedaten auf Schlüssel-Wert-Paare
    - Reduce: Zusammenführung der Zwischenergebnisse
  - Framework
    - Enge Verzahnung mit dem Datenspeichersystem (z. B. Google File System)
    - Scheduling und Verteilung von Worker-Prozessen
    - Fehlerbehandlung bei Worker-Ausfällen
- Kritik an MapReduce
  -  [Michael Stonebraker, Daniel J. Abadi, David J. DeWitt et al. \*\*MapReduce and parallel DBMSs: Friends or foes?\*\* \*Communications of the ACM\*, 53\(1\):64–71, 2010.](#)

