

# Programmierunterstützung im Kontext von Cloud Computing

Google MapReduce, Disco und BOOM

Alexander Singer  
AlexSinger@gmx.net

Seminar  
Ausgewählte Kapitel der Systemsoftware: Cloud Computing

17. Juni 2010

# Motivation

## Problem

- Ständig wachsende Datenmenge
- Trotzdem kurze Bearbeitungsdauer gewünscht

# Motivation

## Problem

- Ständig wachsende Datenmenge
- Trotzdem kurze Bearbeitungsdauer gewünscht

## Lösung

Verarbeitung in Rechner-Clustern

## Neue Fragen

- Wie die große Anzahl Rechner nutzen?
- Was passiert beim Ausfall einzelner Rechner?
- Welche Rolle spielt das Speichersystem?

# Motivation

## Probleme durch Cluster

- Programmausführung muss parallelisierbar sein
  - Hohe Ausfallwahrscheinlichkeit
  - Netzwerklast durch Datentransfer
  - Fehlende Programmierunterstützung
- ⇒ Komplexe Programme für simple Berechnungen

## Cloud Computing

⇒ Größerer Nutzerkreis durch „mietbare“ Cluster

# Motivation

## Wunsch: „datenzentrisches Programmieren“

- Daten stehen im Mittelpunkt
- Keine Koordination von Prozessen/Rechnern

# Motivation

## Wunsch: „datenzentrisches Programmieren“

- Daten stehen im Mittelpunkt
- Keine Koordination von Prozessen/Rechnern

## MapReduce-Konzept

Bietet Möglichkeit zum datenzentrischen Programmieren

# Übersicht

- 1 Google MapReduce
  - Implementierung
  - Beispiele
  - Ausfallbehandlung
  - Optimierungen und Performanz
- 2 Disco
  - Disco MapReduce
  - Disco Distributed Filesystem
  - Discodex
- 3 BOOM
  - Ziel des BOOM-Projekts
  - BOOM MapReduce
  - BOOM File System
- 4 Fazit

# Übersicht

- 1 Google MapReduce
  - Implementierung
  - Beispiele
  - Ausfallbehandlung
  - Optimierungen und Performanz
- 2 Disco
  - Disco MapReduce
  - Disco Distributed Filesystem
  - Discodex
- 3 BOOM
  - Ziel des BOOM-Projekts
  - BOOM MapReduce
  - BOOM File System
- 4 Fazit

# Google MapReduce

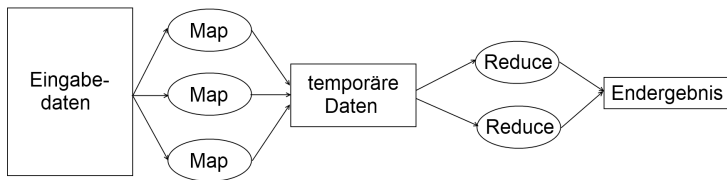
## Google MapReduce Framework

### Aufteilung der Datenverarbeitung

- Map: Abbildung der Daten auf Schlüssel-Wert-Paare
- Reduce: Zusammenführen der Paare zu Endergebnis

### Framework übernimmt:

- Koordination der Rechner
- Ausfallbehandlung
- Verteilung der Daten



# Implementierung des MapReduce-Frameworks

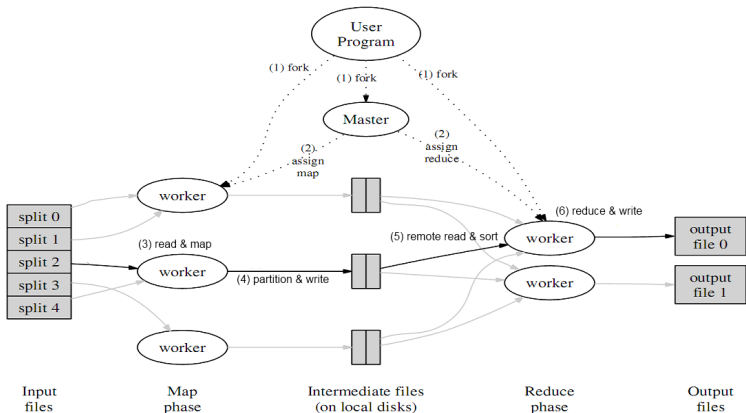
## Master

- Eine einzige Instanz pro MapReduce-Operation
- Teilt Aufgabe zu
- Überwacht Worker-Verhalten
- Informiert Reduce-Worker über Zwischenergebnisse

## Worker

- Beliebig viele Instanzen
- Erhalten Aufgabe und Daten zugewiesen

# Implementierung und Ablauf



aus Google MapReduce Paper [1] (leicht abgeändert)

# Beispiel: Wörter zählen

## Annahmen

- 3 Map-Worker
- 2 Reduce-Worker

## Beispiel-Eingabedaten

„Here, each document is split in words, and each word is counted”

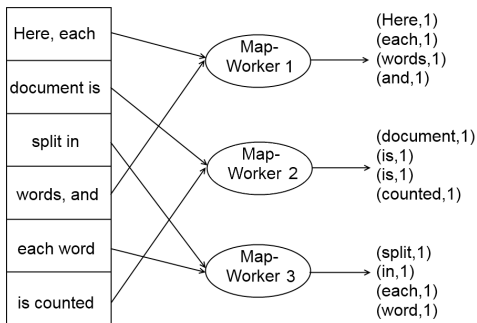
## Schritte 1 und 2

- Erstellen des Masters
- Erstellen der Worker
- Map- und Reduce-Funktionen zuweisen

# Beispiel: Wörter zählen

## Schritt 3: Map-Funktion

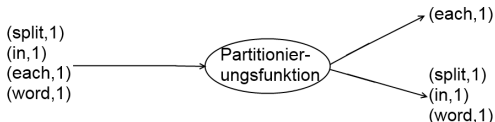
- Daten werden aufgeteilt
- Jeder Map-Prozess erhält mehrere Teile
- Abbildung auf Schlüssel-Wert-Paare



# Beispiel: Wörter zählen

## Schritt 4: Partitionierung

- Aufteilung der Daten für Reduce-Prozess
- Wird von jedem Map-Prozess durchgeführt



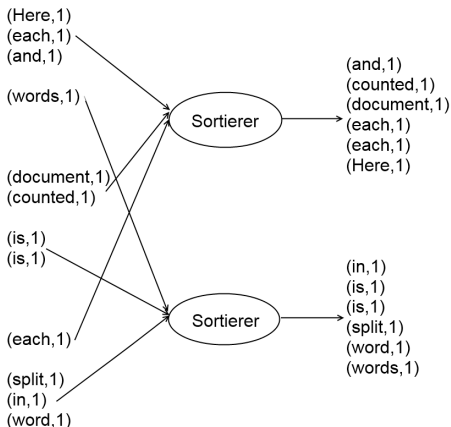
## Partitionierungsfunktion

- Standardmäßig: Hash(Schlüssel) modulo #Reduce-Worker
- Hier: Verteilung anhand des Anfangsbuchstabens

# Beispiel: Wörter zählen

## Schritt 5: Lesen und Sortieren der Zwischenergebnisse

- Von jedem Reduce-Prozess ausgeführt
- Lesen partitionierter Zwischenergebnisse per Fernaufruf



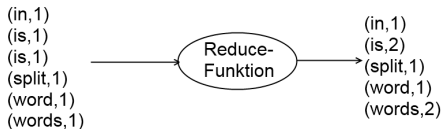
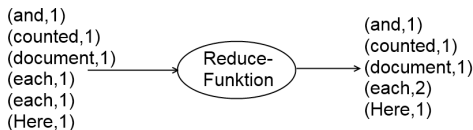
### Sortieren

- Partitionierte Zwischenergebnisse werden sortiert
- Weiterleitung sortierter Ergebnisse an Reduce-Funktion

# Beispiel: Wörter zählen

## Schritt 6: Reduce-Funktion

- Wird für jeden Schlüssel aufgerufen
- Erhält Schlüssel und alle zugehörigen Werte
- Ausgabe ist Endergebnis



# Weitere Beispiele

## Verteiltes grep

- Map-Funktion
  - Sortiert Unzutreffendes aus
  - Speichert Treffer als Zwischenergebnisse
- Reduce-Funktion
  - Leitet Zwischenergebnisse 1:1 durch
  - Aber: Andere Ergebnisreihenfolge durch Sortierung

# Weitere Beispiele

## Indexerstellung

- Map-Funktion
  - Eingabedaten: Dokumente, Webseiten ...
  - Ausgabe: Paare aus Schlagwort und Dokument-ID
- Reduce-Funktion
  - Eingabe: Schlagwort und Iterator über Dokument-IDs
  - Ausgabe: Schlagwort und Liste mit Dokument-IDs

# Ausfallbehandlung

## Ausfallbehandlung ist extrem wichtig

Durch große Anzahl „Commodity“-Maschinen hohe Ausfallwahrscheinlichkeit

## Zwei Arten von Ausfällen

- Worker-Ausfälle
  - Master fragt regelmäßig Zustand ab
  - Betroffene Tasks werden zurückgesetzt
  - Eventuell Wiederholung des Map-Tasks
- Master-Ausfall
  - Kommt sehr selten vor
  - Google Entscheidung: keine Recovery
  - ⇒ MapReduce-Operation wird wiederholt

# Optimierungen: Partitionierung

## Partitionierungsfunktion

Aufteilung der Zwischenergebnisse in verschiedene „Buckets“

- Aufteilung der Zwischenergebnisse anhand des Schlüssels
- Im Normalfall Hashfunktion und Modulo
- Gleiche Schlüssel an gleichen Reduce-Worker
- Eigene Implementierung möglich

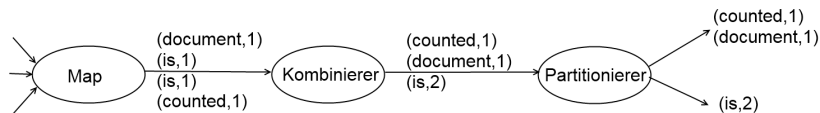
## Partitionierungsfunktion

Hauptgrund für Parallelisierbarkeit der Reduce-Phase

# Optimierungen: Kombinerer

## Kombinerer-Funktion

- Reduktion des Zwischenergebnisses
- ⇒ Kann Zahl der Ein-/Ausgabeoperationen senken



# Optimierungen: Backup-Tasks

## Problem

Ausfall, Verzögerung einzelner Rechner

## Lösung: Backup-Tasks

„Klone“ noch laufender Tasks

- Gegen Ende der MapReduce-Operation eingesetzt
  - Duplizierung von laufenden Tasks
  - Ergebnis von Klon **oder** Original
- ⇒ Verzögerungen werden eingedämmt
- Angabe von Google: ohne Backup-Tasks bis zu 44% langsamer [1]

# Optimierungen: Einsatz des Google File Systems

## Probleme durch große Datenmengen

- Festplattengeschwindigkeit wird wichtig
- Netzwerkbandbreite wird knapp

## Lösung: Einsatz des Google File Systems

- Verteilung der Daten
- Optimierung auf große Daten
- Optimierung auf Lesen/Anhängen von/an Dateien
- Lesen von mehreren Platten gleichzeitig

# Optimierungen: Einsatz des Google File Systems

## Google File System

Speichersystem ist Grundlage für MapReduce-Performanz

- Hoher Gesamtdurchsatz der Festplatten
- Geschickte Verteilung der Worker-Prozesse
- Abstimmung von Chunk-Größe und Split-Größe

# Performanz

Keine Vergleiche zu „alten“ Google-Programmen

## Kritik an MapReduce nach Vergleich mit DBMS [2]

- Kein Schema
- Keine Hilfsstrukturen (B-Bäume, Indices)
- Berechnungen 2x bis 6x schneller mit parallelem DBMS
- Bis zu 10 mal größeres Cluster nötig um DBMS zu erreichen

## Allerdings

- MapReduce-Performanz oft ausreichend
- Schnelleres Laden von Daten
- DBMS Installation/Konfiguration sehr viel aufwendiger

## 1 Google MapReduce

- Implementierung
- Beispiele
- Ausfallbehandlung
- Optimierungen und Performanz

## 2 Disco

- Disco MapReduce
- Disco Distributed Filesystem
- Discodex

## 3 BOOM

- Ziel des BOOM-Projekts
- BOOM MapReduce
- BOOM File System

## 4 Fazit

# Disco

## Das Disco Projekt

- Geschrieben in Erlang
- Benutzerfunktionen standardmäßig in Python
- Begonnen vom Nokia Research Center
- Open-Source

## Komponenten von Disco

- Disco MapReduce: Implementierung von Googles MapReduce Konzept
- Disco Distributed Filesystem: verteiltes Speichersystem nach Google-Konzept
- Discodex: Indexsystem zum schnellen Datenzugriff

# Disco MapReduce

## Open-Source-Implementierung des Google MapReduce-Konzeptes

- Master überwacht Worker
- Worker führen Map- und Reduce-Tasks aus

## Unterschied zum Google-Konzept

- Zusätzlicher globaler Koordinator
  - Genauere Bestimmung der Partitionierung
- ⇒ Freiere Verkettung vom Tasks

# Disco Distributed Filesystem

## Disco Distributed Filesystem

Orientiert sich stark am Google File System

- Master-Chunkserver-Konzept
- Replizierung der Daten
- Ausgelegt für Lese/Anhänge-Operationen mit großen Dateien

## Zusammenspiel mit MapReduce

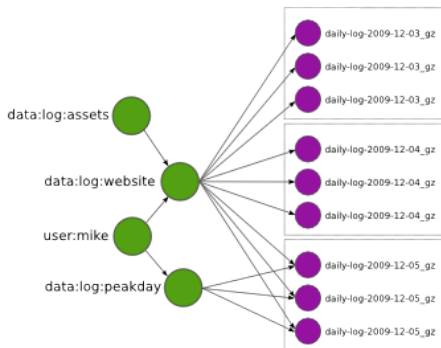
Auch Disco MapReduce benötigt verteiltes Dateisystem

- Netzwerklast
- Festplattendurchsatz

# Disco Distributed Filesystem

## Disco: Tag-based File System

- „Blobs“: Daten im Dateisystem
- „Tags“: Metadaten über Blobs



(aus [3])

## Vorteile

- Graphenstruktur aus Tags
- Bietet selektiven Zugriff

# Discodex

## Discodb-Objekte

Im Disco Distributed Filesystem gespeicherte Indexdateien

## Discodex

Mit MapReduce erstellter Index für schnellen Datenzugriff

- Erstellt Index aus Schlüssel-Wert-Paaren
- Schlüssel-Wert-Paare liegen in Discodb-Objekten
- Minimales perfektes Hashing
- Wertezugriff in konstanter Zeit  $O(1)$

# Übersicht

- 1 Google MapReduce
  - Implementierung
  - Beispiele
  - Ausfallbehandlung
  - Optimierungen und Performanz
- 2 Disco
  - Disco MapReduce
  - Disco Distributed Filesystem
  - Discodex
- 3 **BOOM**
  - Ziel des BOOM-Projekts
  - BOOM MapReduce
  - BOOM File System
- 4 Fazit

# BOOM

## BOOM (Berkeley Orders Of Magnitude)

Ziel: vielfach größere/komplexere Systeme in Bruchteil von Code

## BOOM-Sicht

Programmablauf = Zustand und Zustandsmodifikation

# BOOM

## BOOM-Zitat [4]

- „Distributed computing is all about state“
- „Computing = Creating, updating, and communicating that state“

## BOOM Analytics

- Zum Testen dieses Ansatzes
- Portierung von Hadoop MapReduce
- Reimplementierung des Hadoop File Systems (HDFS)
- Implementierung mit Relationen und Overlog

# BOOM

## Overlog

Overlog ist Erweiterung von Datalog

- Modifikation von Daten
- Ortsbestimmung von Daten
- SQL-artige Erweiterungen (Primärschlüssel, Aggregate)

## Overlog-Zeitschritt

Atomarer lokaler Abschnitt

- Nur lokaler Zustand sichtbar
- ⇒ Keine Unterbrechung von Operationen
- ⇒ Konsistenter Zustand am Ende
- Unterteilbar in drei Phasen

# BOOM

## Event

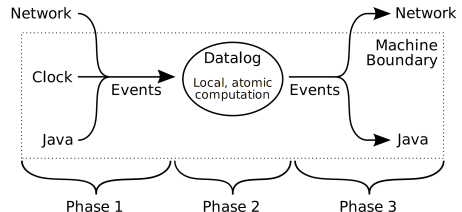
Schnittstelle zum Rest des Systems

- Java-Komponenten
- Netzwerk
- Zeitgeber

# BOOM

## Die 3 Phasen des Zeitschrittes

- Phase 1:
  - Konvertierung von Event in Tupelmodifikation
- Phase 2:
  - Ausführung Tupelmodifikation
  - Ausführung Overlog-Regeln bis Fixpunkt
- Phase 3:
  - Speicherung neuer Daten
  - Absetzen von Events



aus [4]

# BOOM MapReduce

## BOOM MapReduce

Portierung von Hadoop-MapReduce-Kern nach Overlog

### Ziele

- Einfacherer, kürzerer Code
- Höhere Erweiterbarkeit
- LATE-Erweiterung zeigt: Ziel erreicht
  - Neue Scheduler-Strategie nur wenig Code
  - Einfügen in bestehenden Kern sehr einfach

# BOOM File System

## BOOM File System: Reimplementierung des Hadoop File Systems

- Overlog für Metadatenbearbeitung
- Java für Datentransfer

## Herausforderung: verteilte Speicherung

- Metadaten auf Master
- Daten auf Speicherknoten

## Ergebnis

- ⇒ Codelänge auf <10%
- ⇒ „hot standby“-Erweiterung nur 53 Zeilen Code

# Fazit

## MapReduce

- Starke Unterstützung
- Open-Source-Implementierungen
- Einfach zu verwenden

MapReduce braucht verteiltes Speichersystem

# Fragen?

**Noch Fragen?**

# Referenzen



Jeffrey Dean and Sanjay Ghemawat.

MapReduce: Simplified Data Processing on Large Clusters, 2004.



Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. Dewitt, Samuel Madden, and Michael Stonebraker.

A Comparison of Approaches to Large-Scale Data Analysis.

In *SIGMOD '09: Proceedings of the 2009 ACM SIGMOD International Conference*. ACM, June 2009.



Disco.

Disco Distributed Filesystem.

<http://discoproject.org/doc/start/ddfs.html> (Stand 02.06.2010).



Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell C Sears.

BOOM: Data-Centric Programming in the Datacenter.

Technical Report UCB/EECS-2009-113, EECS Department, University of California, Berkeley, August 2009.