MapReduce in der Praxis

Rolf Daniel

Seminar Multicore Programmierung

09.12.2010

Agenda

Motivation

Large Scale Data Processing

- Verarbeitung von immer größer werdenden Datenmengen
- In möglichst kurzer Zeit
- Unter Verwenudng zahlreicher CPUs
- Soll sehr einfach funktionieren
  $\Rightarrow$ Parallele Verarbeitung der Daten in Rechner–Clustern

Motivation

Abbildung: Google rack mit 40 Servern[1]

[1] Quelle: http://news.cnet.com

Motivation

Neue Probleme

- Verarbeitung der Daten muss parallelisierbar sein

Motivation

Neue Probleme

- Verarbeitung der Daten muss parallelisierbar sein
- Viele Knoten führen zu einer hohen Ausfallrate

Motivation

Neue Probleme

- Verarbeitung der Daten muss parallelisierbar sein
- Viele Knoten führen zu einer hohen Ausfallrate
- Hohe Netzwerklast durch Verteilung der Daten

Motivation

Neue Probleme

- Verarbeitung der Daten muss parallelisierbar sein
- Viele Knoten führen zu einer hohen Ausfallrate
- Hohe Netzwerklast durch Verteilung der Daten
- Komplexität für Programme steigt

Motivation

MapReduce–Konzept

MapReduce bietet:

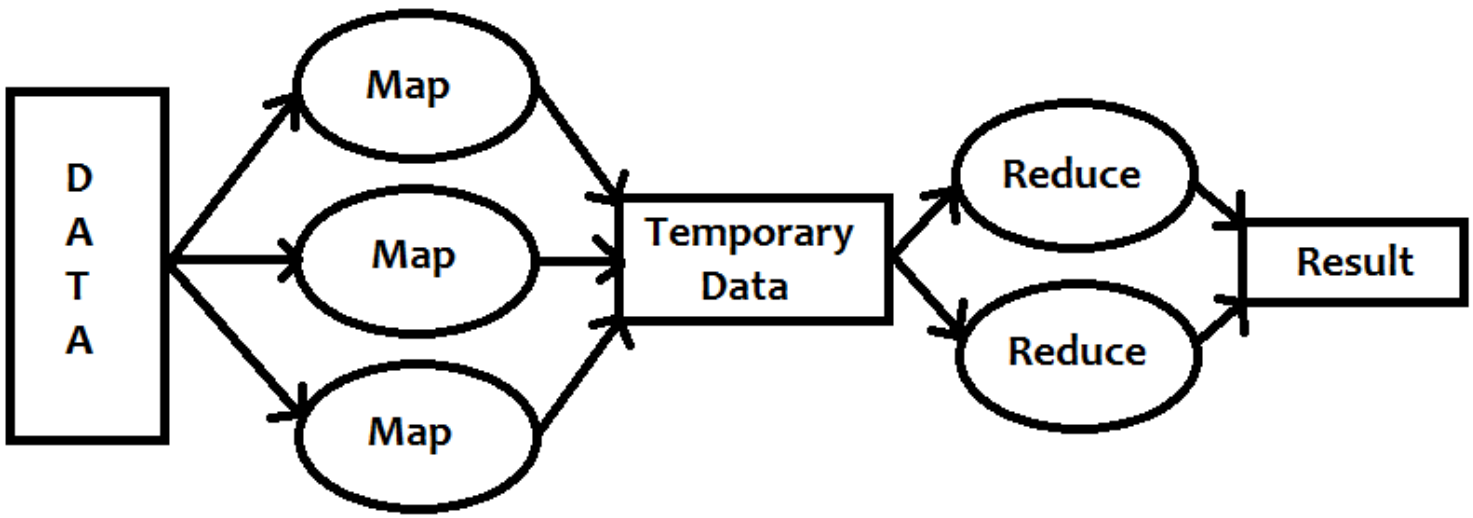
- Automatische Parallelisierung und Verteilung
- Fehlertoleranz
- Abstraktion für Programmierer
- Statusüberwachung

Programmiermodell

MapReduce–Berechnungen

Die Berechnung wird durch zwei Funktionen beschrieben

- Map: bildet Eingabedaten auf Schlüssel/Wert–Paare ab
- Reduce: fügt diese Paare zum Endergebnis zusammen



Programmiermodell

MapReduce Framework

Das Framework ist zuständig für:

- automatische Parallelisierung und Verteilung
- Fehlerbehandlung
- Scheduling der Kommunikation zwischen den Rechnern

GFS

Google File System

- Eingabedaten werden in Blöcke aufgeteilt
- Blöcke werden kopiert und auf verschiedenen Rechnern gespeichert
- Für MapReduce–Berechnungen werden die meisten Daten lokal gelesen
  $\Rightarrow$ Netzwerklast wird reduziert

Implementierung des Frameworks

Aufbau des Clusters (Stand 2004):

- mehrere hundert/tausend Maschinen
  2–CPU x86, 2–4 GB Arbeitsspeicher, Linux
- Standardnetzwerkhardware
  100 Mb/s oder 1 Gb/s Ethernet
- Speicherung von Daten lokal auf IDE Festplatten
- GFS zur Verwaltung der Daten
- Scheduling–System

Implementierung des Frameworks

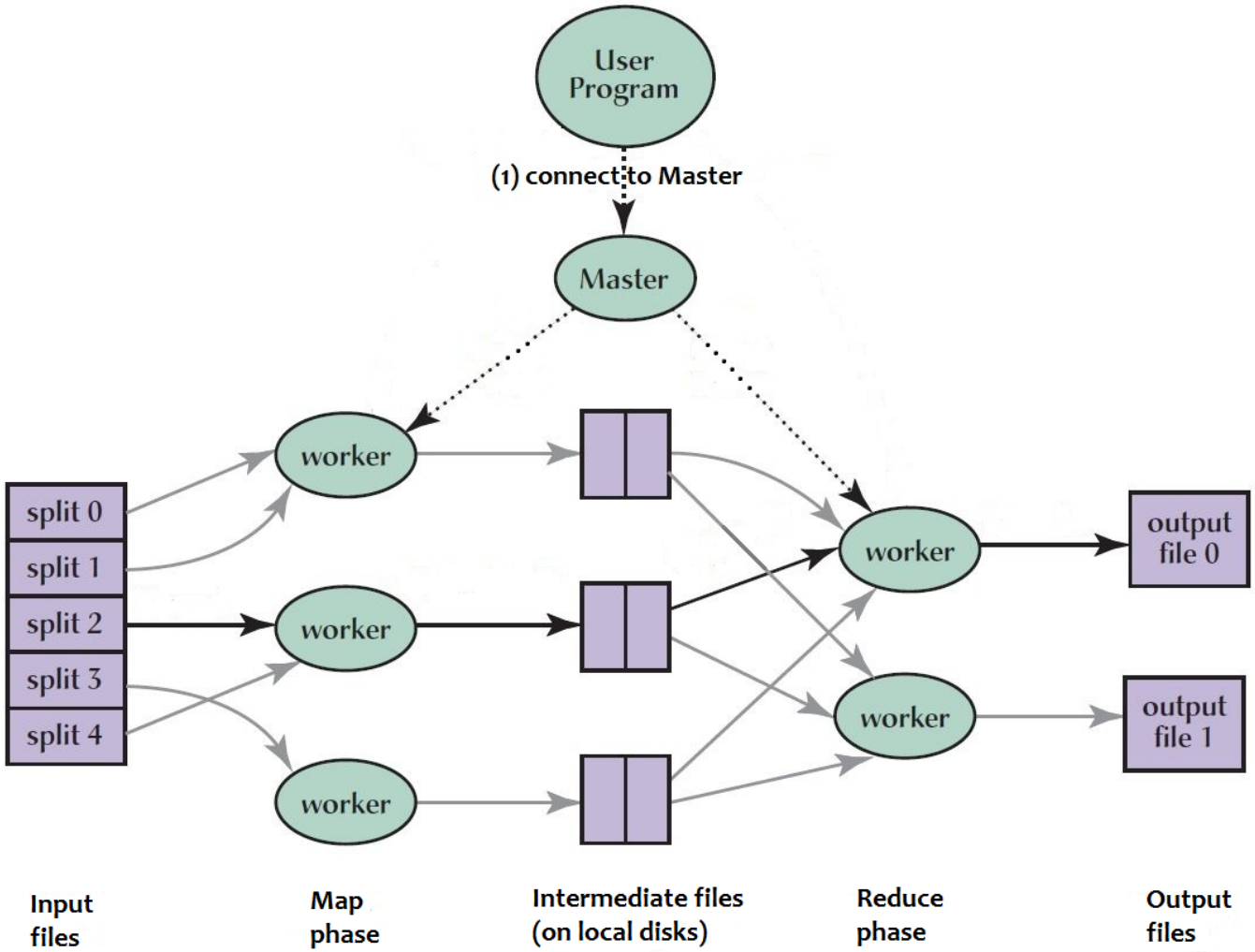
Master

- Es gibt nur einen pro MapReduce–Operation
- Weist den Workern Aufgaben zu
- Speichert Zustände der Worker (*idle*, *in–progress*, *completed*)
- Informiert Reduce–Worker über Zwischenergebnisse

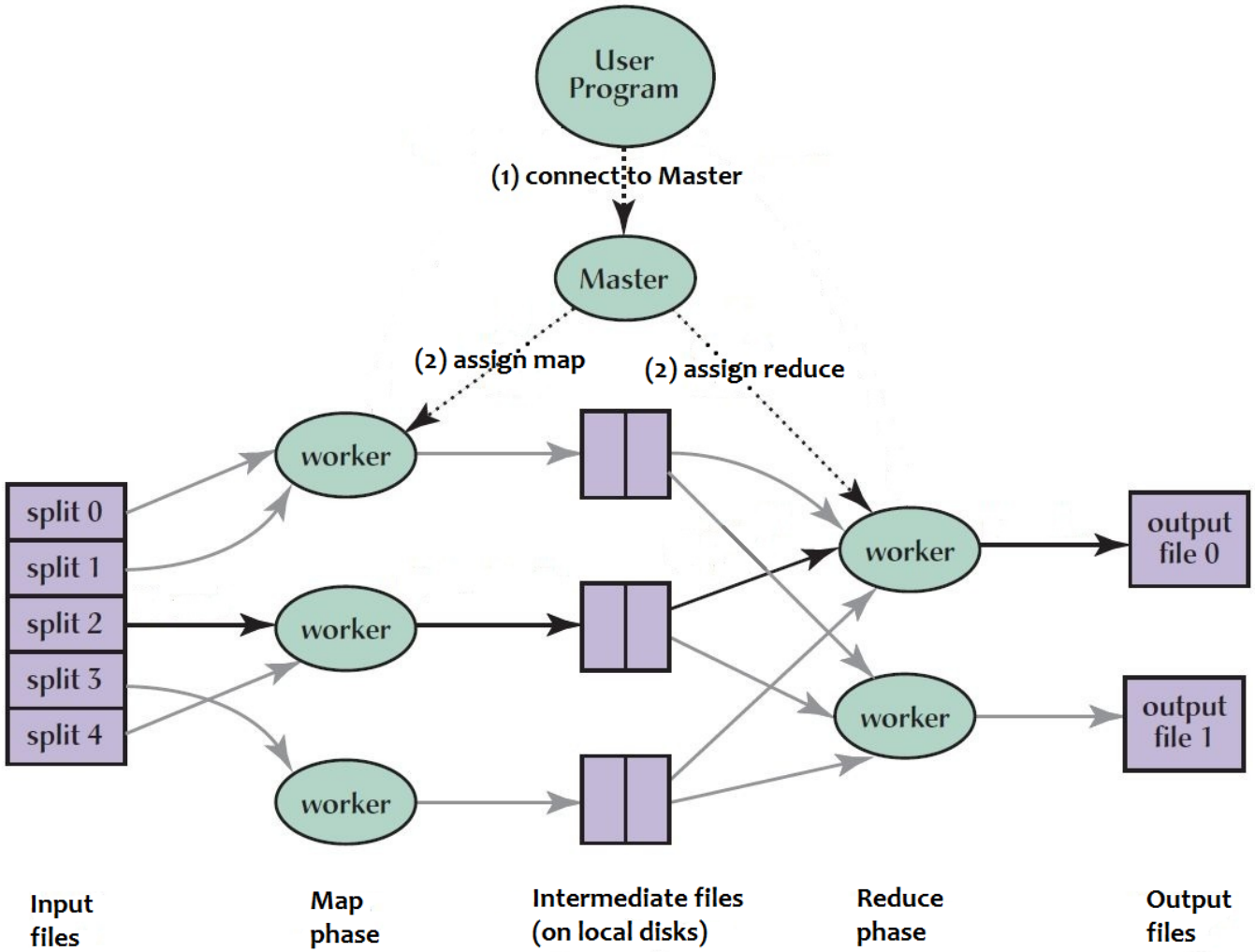
Worker

- Anzahl ist beliebig
- Bekommen Aufgaben und Daten zugewiesen
- Führen Map– bzw. Reduce–Tasks durch

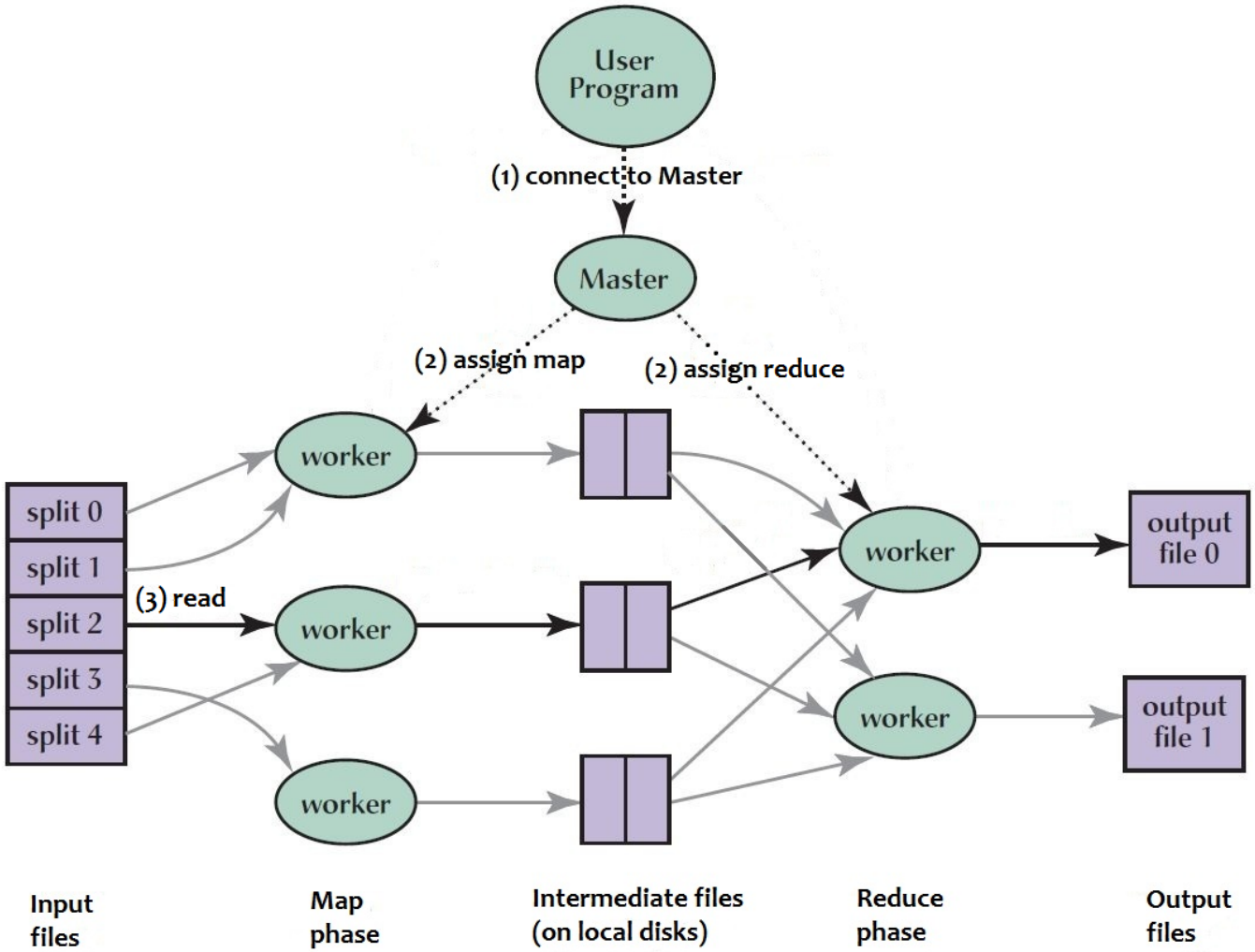
Ablauf


User Program
(1) connect to Master
Master
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
worker
worker
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

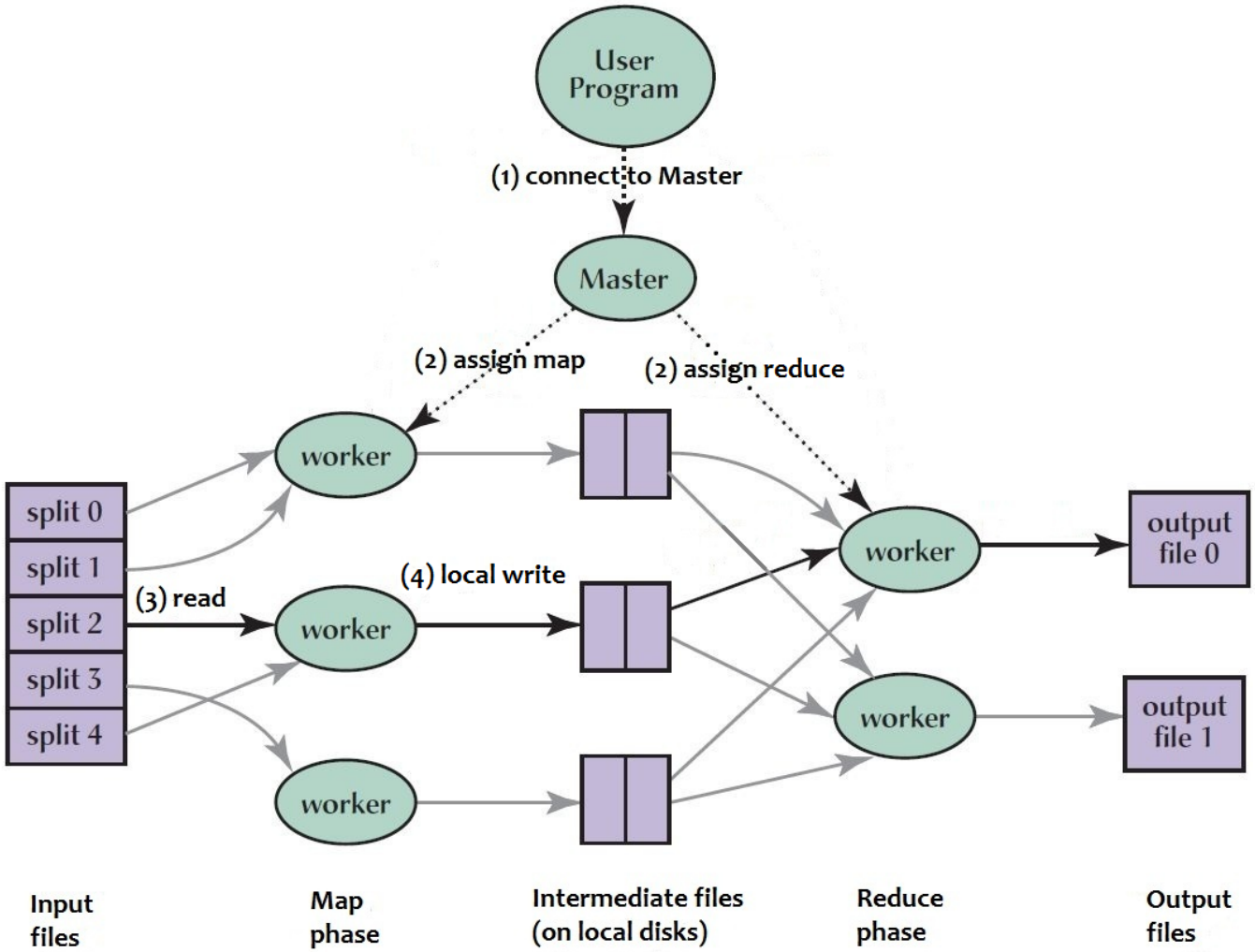
Ablauf


User Program
(1) connect to Master
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

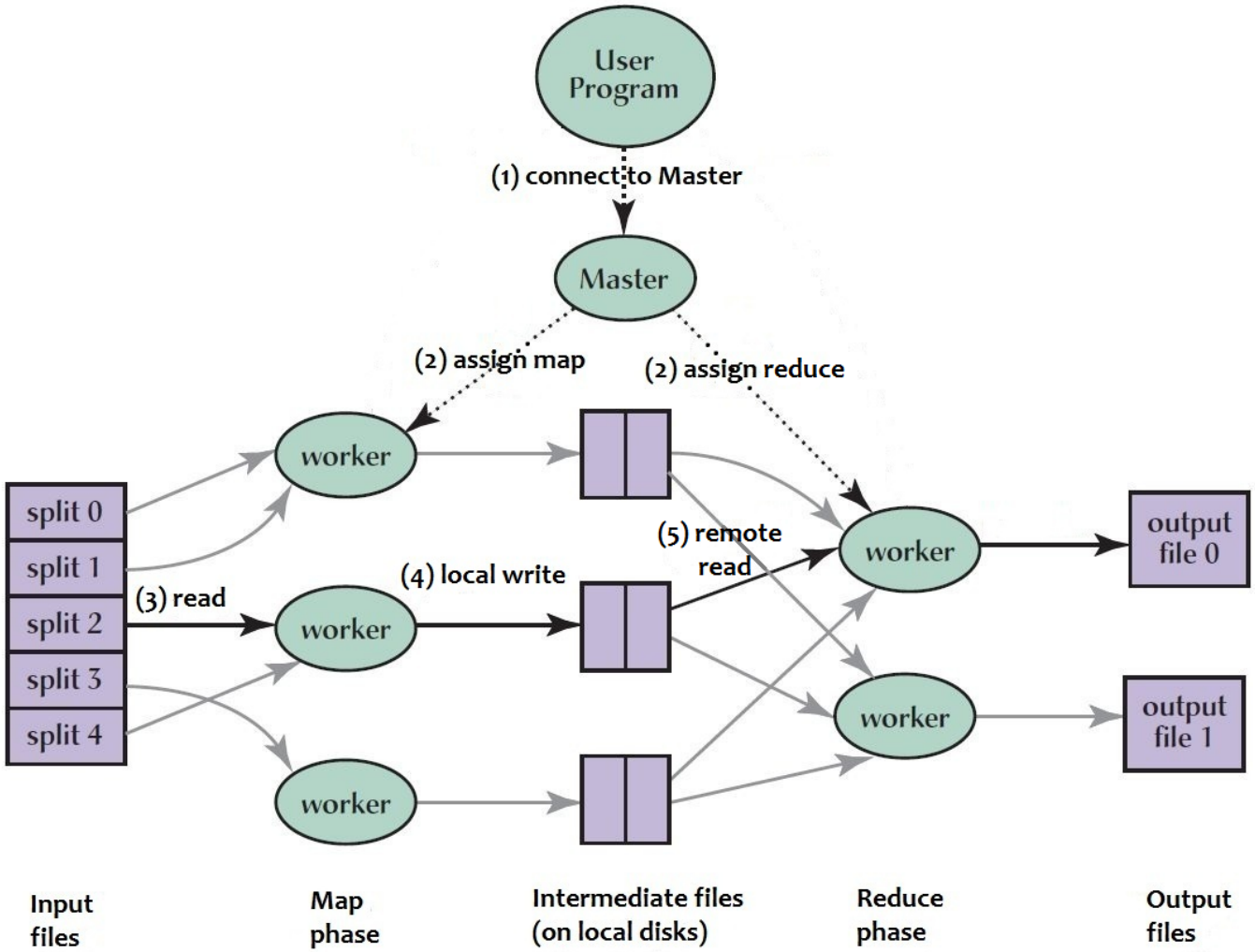
Ablauf


User Program
(1) connect to Master
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

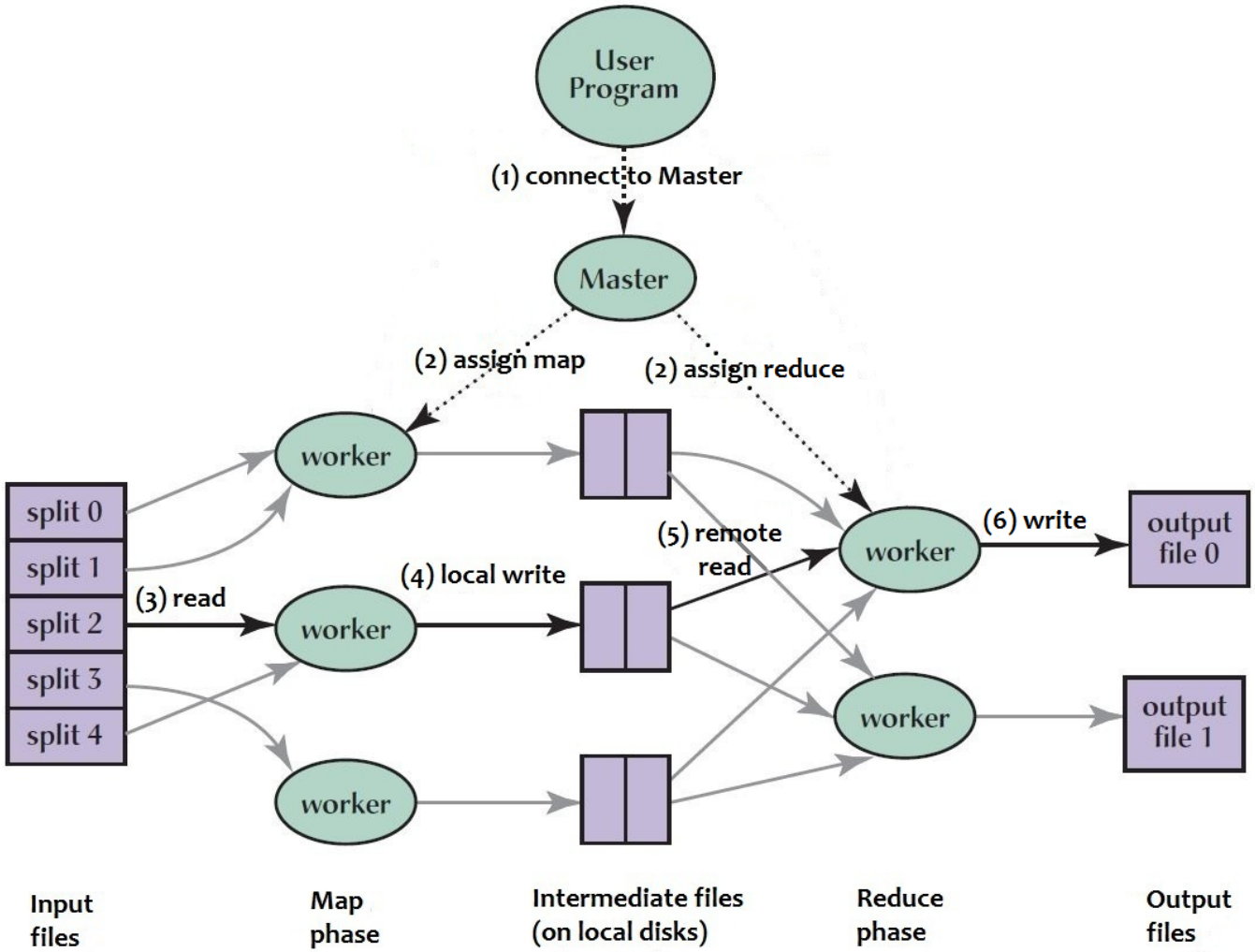
Ablauf


User Program
(1) connect to Master
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Ablauf


User Program
(1) connect to Master
Master
(2) assign map
(2) assign reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Ablauf


User Program
(1) connect to Master
Master
(2) assign map
(2) assign reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Wörter in einem Dokument zählen

Eingabedaten: Text

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

Wörter in einem Dokument zählen

Map–Funktion definieren

```
map(String key, String value):
  // key: number of the part of a sentence
  // value: content of the part of a sentence
  for each word w in value:
    EmitIntermediate(w,"1");
```

Wörter in einem Dokument zählen

Reduce–Funktion definieren

```
reduce(String key, Iterator values):
  // key: word
  // values: an iterator of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Wörter in einem Dokument zählen

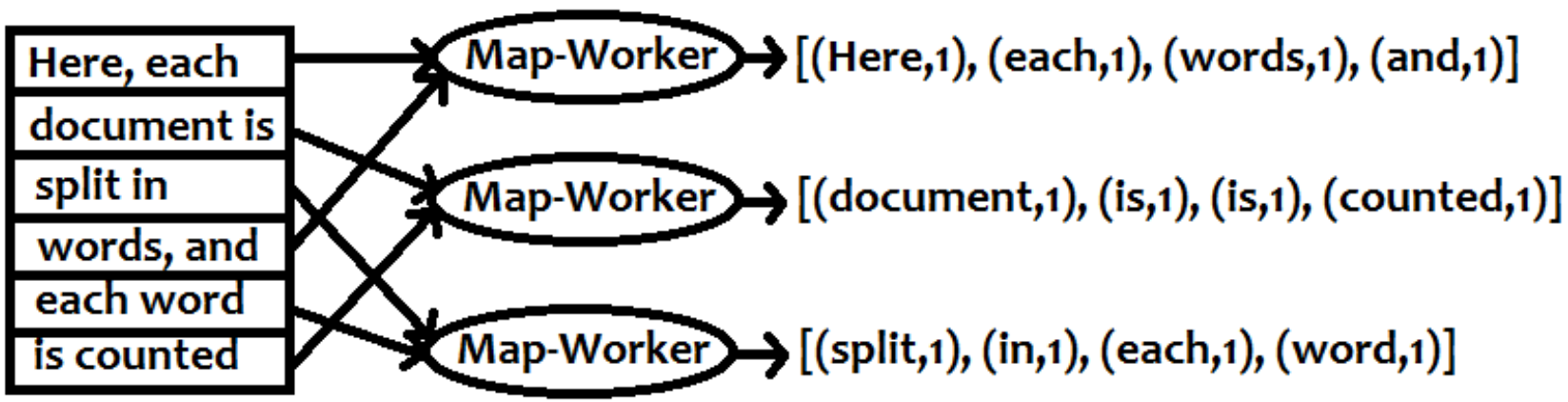
Schritt 1

- Eingabedaten in mehrere Satzteile zerlegen
- Jeder Satzteil erhält eine Nummer
  $\Rightarrow$ Liste aus Schlüssel/Wert–Paaren
- Verbindung zum Master herstellen

Wörter in einem Dokument zählen

Schritt 2 und 3

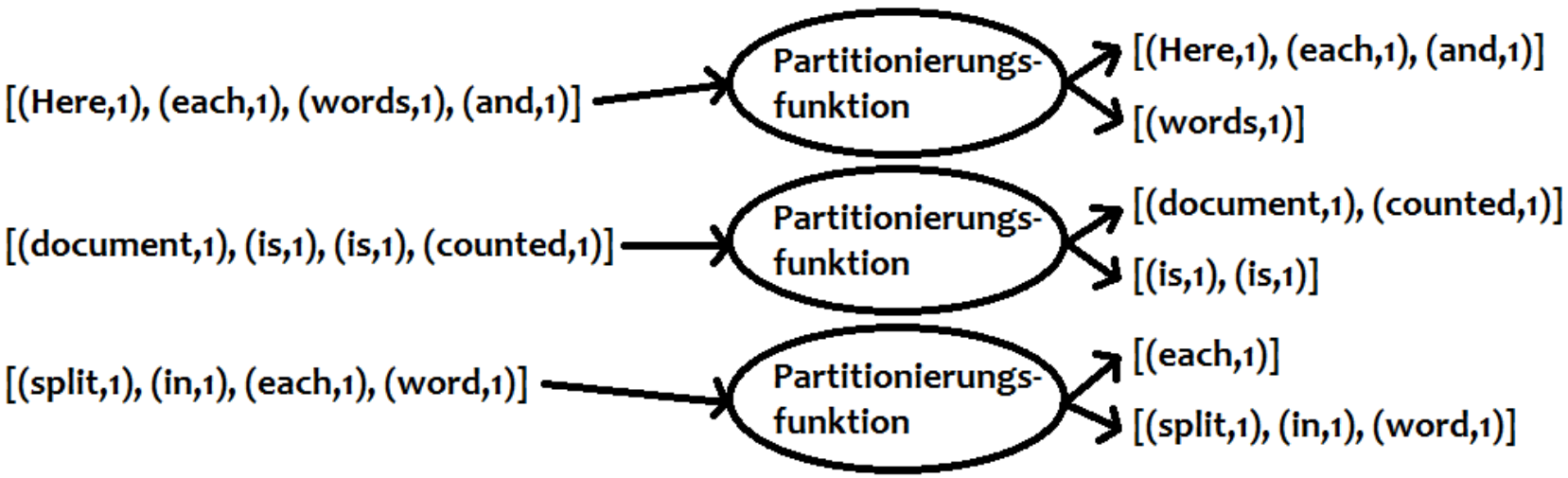
- Master weist Workern je einen Task zu
- Jeder Map–Worker erhält mehrere Satzteile
- Map–Worker bilden Daten auf Schlüssel/Wert–Paare ab
- Diese Zwischenergebnisse werden gepuffert



Wörter in einem Dokument zählen

Schritt 4

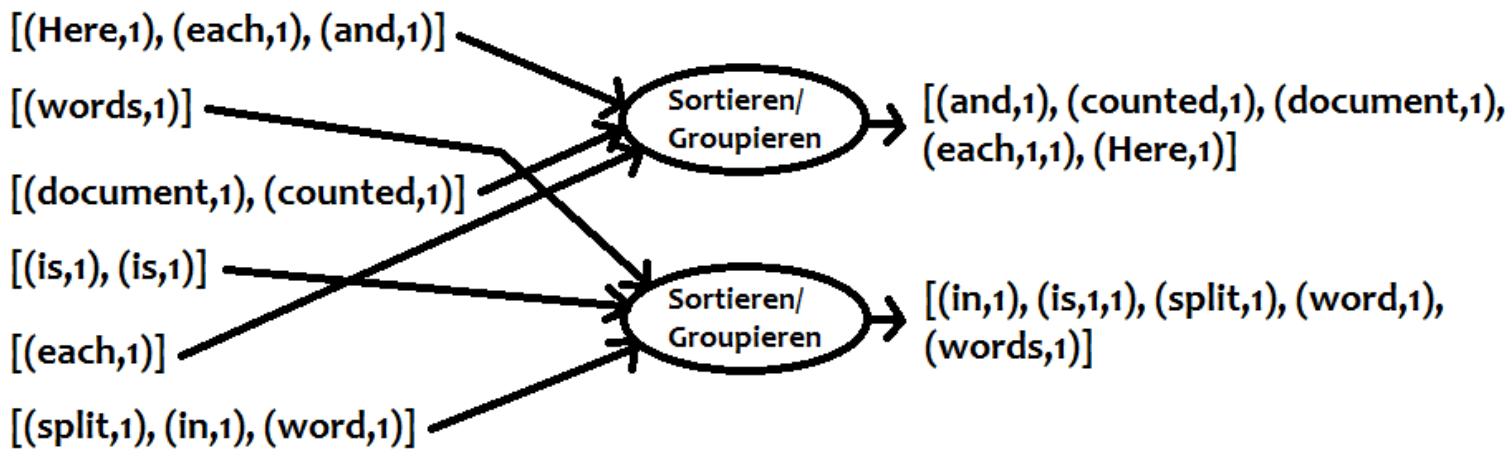
- Gepufferte Daten werden periodisch
  - auf lokale Festplatte geschrieben und
  - für die Reduce–Worker partitioniert
- Speicherort wird Master mitgeteilt



Wörter in einem Dokument zählen

Schritt 5

- Master teilt Reduce–Workern Speicherort mit
- Reduce–Worker
  - lesen partitionierte Zwischenergebnisse per RPC und
  - sortieren diese anhand des Schlüssels

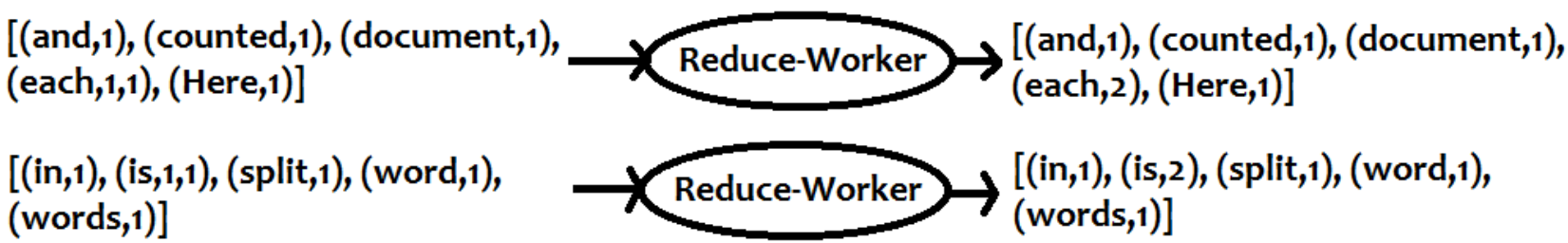


Wörter in einem Dokument zählen

Schritt 6

Wird von jedem Reduce–Worker durchgeführt:

- Iteriert über sortierte Daten
- Für jeden Schlüssel und die zugehörigen Werte wird Reduce–Funktion aufgerufen
- Ausgaben aller Reduce–Funktionen werden zu Endergebnis konkateniert



Weitere Beispiele

Invertierter Index

- 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

Weitere Beispiele

- Verteiltes Grep
- Häufigkeit von URL–Zugriffen
- Term–Vektor pro Host
- Verteiltes Sortieren
- ...

Ausfallbehandlung

Ausfall eines Workers

- Master fragt periodisch Zustände ab
- Betroffene Tasks werden zurückgesetzt und erneut ausgeführt
- Map–Worker:
  - alle fertiggestellten und in Bearbeitung befindlichen Tasks werden zurückgesetzt
- Reduce–Worker:
  - nur in Bearbeitung befindliche Tasks werden zurückgesetzt

Ausfallbehandlung

Ausfall des Masters

- Kommt sehr selten vor
- Keine Recovery
  $\Rightarrow$ MapReduce–Operation wird wiederholt

Disco

Das Disco Projekt

- Open–Source Implementierung in Erlang
- Entwickelt von *Nokia Research Center*
- Basiert auf einer Master–Slave–Architektur
- Verteiltes Filesystem
- Discodex (Index)

Architektur

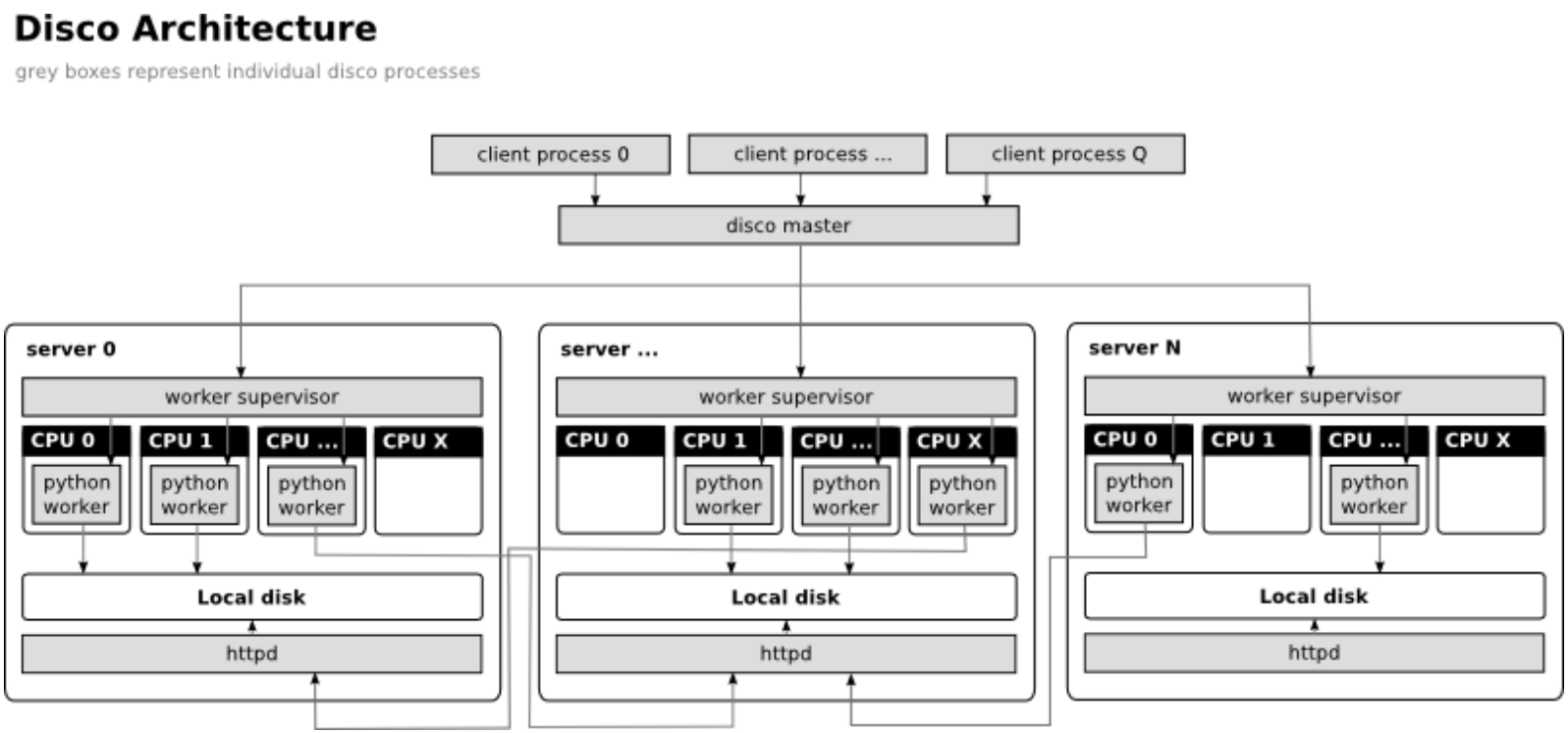


Abbildung: Disco–Architektur[2]

[2] Quelle: http://discoproject.org

Discos MapReduce

Discos MapReduce

- Master ist für Verteilung der Jobs zuständig
- Master erstellt auf jedem Knoten einen Worker Supervisor
- Map– bzw. Reduce–Tasks werden von Python–Workern durchgeführt
- Benötigte Daten werden per HTTP übertragen

DDFS

Disco Distributed File System

- sehr ähnlich zu GFS
- Zuständig für Verteilung, Persistenz, Adressierung und Zugriff der Daten
- Tag–basiertes Filesystem
- Fehlertoleranz und hohe Verfügbarkeit durch Replikation

DDFS

Tag–basiertes Filesystem

- *Blobs*: Können beliebige Objekte (Files) im Filesystem sein
- *Tags*: Beinhalten Metadaten über *Blobs*

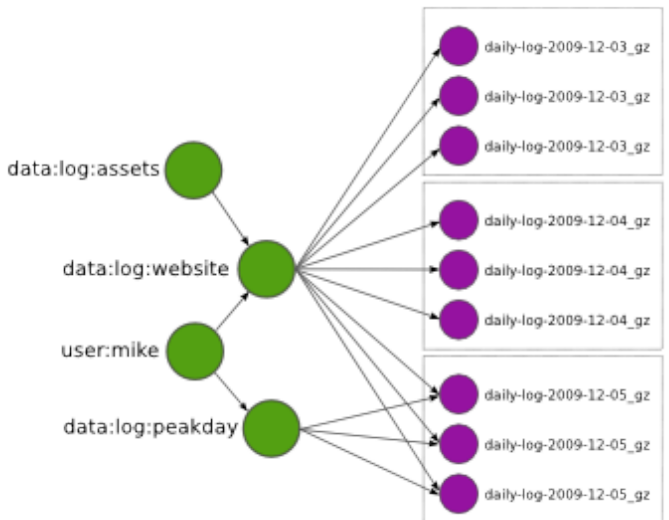


Abbildung: Blob/Tag–Konzept[3]

[3]Quelle: http://discoproject.org

Discodex

Discodex

- Indexierung der Daten für eine konstante Zugriffszeit
- Arbeitet ähnlich wie ein normaler Index
- Indizes werden durch Schlüssel/Werte–Paare repräsentiert
- Indizes werden in *ichunks* aufgeteilt und im DDFS gespeichert
- *Discodb–Objekt*: Datenstruktur speichert Referenz auf die *ichunks*

Hadoop

Apache Hadoop

- Software Framework unterstützt datenintensive und verteilte Applikationen
- Inspiriert von Googles MapReduce und GFS
- Open–Source Java Implementierung
- Verteiltes Filesystem
- Wird von *Facebook*, *IBM*, *Yahoo* u.v.m. genutzt

Master–Slave–Architektur

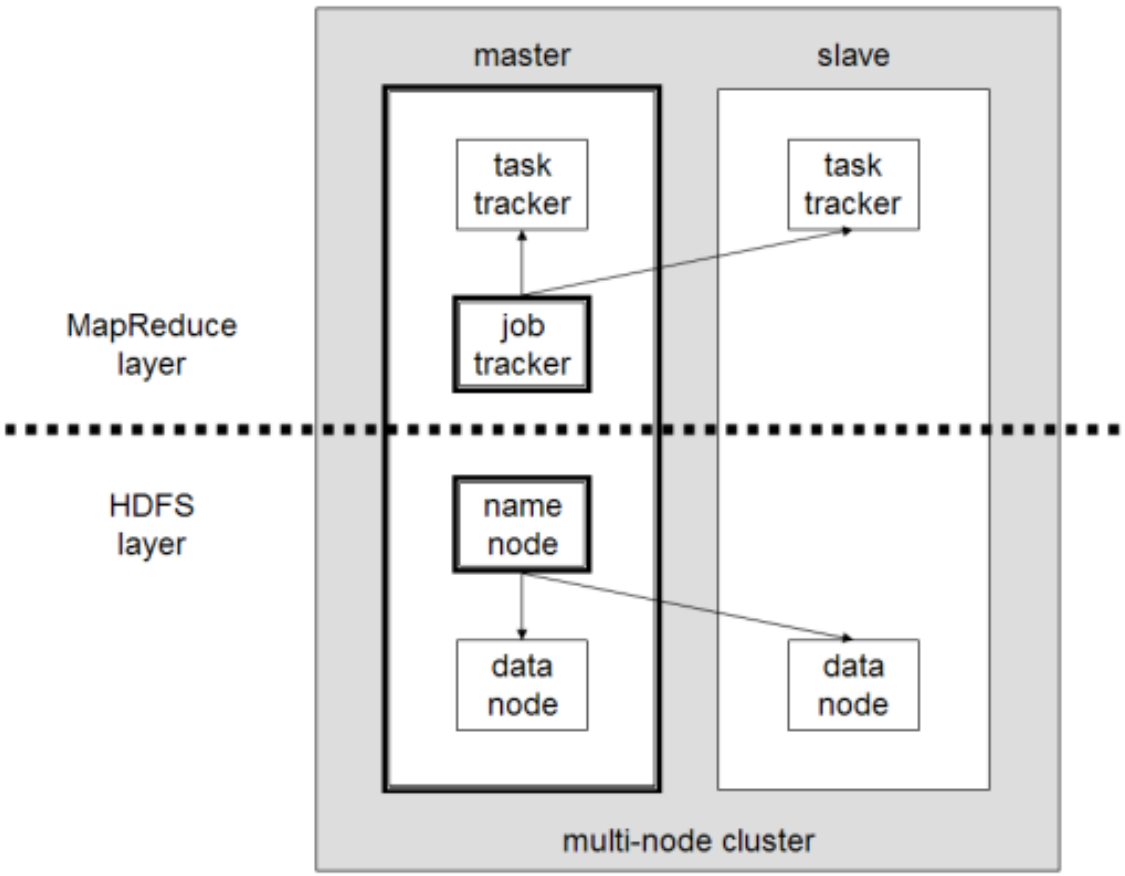


Abbildung: Aufbau eines Hadoop–Clusters[4]

[4]Quelle: http://en.wikipedia.org

Hadoops MapReduce

Hadoops MapReduce

- Ein *JobTracker* steuert mehrere *TaskTracker*
- Ein Job wird in eine Menge von Map– und Reduce–Tasks aufgeteilt
- JobTracker weist TaskTrackern die einzelnen Tasks zu
- TaskTracker hat feste Anzahl an Slots für die Ausführung von Tasks
- *Heartbeat–Protokoll* zwischen JobTracker und TaskTrackern

Hadoops MapReduce

Hadoops MapReduce

- Jeder Map–Task führt Map–Funktion durch, partitioniert Zwischenergebnisse (Schlüssel/Wert–Paare) und speichert sie lokal ab
- JobTracker bildet Reduce–Tasks und weist sie TaskTrackern zu
- Jeder Reduce–Task
  - holt sich die entsprechenden Partitionen
  - sortiert sie lokal nach ihren Schlüsseln
  - führt Reduce–Funktion durch
  - speichert das Ergebnis im verteilten Filsystem

HDFS

Hadoop Distributed File System

- zentraler *NameNode* speichert Metadaten des Filesystems
- Daten werden in Datenblöcke aufgeteilt, repliziert und auf *DataNodes* im Cluster verteilt
- *DataNodes* schicken Heartbeat–Nachrichten zum *NameNode*

BOOM–Projekt

Berkeley Orders of Magnitude

„We seek to build OOM[a] bigger systems with OOM less effort“[b]

---

[a] OOM: Orders Of Magnitude
[b] Quelle: http://boom.cs.berkeley.edu/

BOOM Analytics

- Portierung von Hadoops MapReduce und Neuimplementierung von HDFS
- Geschrieben in Overlog und Java
- Kompaktere und leicht erweiterbare Codebasis

System	**Lines in LATE–Patch**	**Files Modified by LATE–Patch**	**LOC**
Hadoop	2.102	17	6.573
HDFS	–	–	≈21.700
BOOM–MR	82	2	396 + 1.269
BOOM–FS	–	–	469 + 1.431

Fazit

MapReduce

- Einfach zu verwenden
- Open–Source–Implementierungen vorhanden
- Gute Abstraktion für Entwickler

Verteiltes Filesystem wird benötigt!

Noch Fragen?

Vielen Dank für eure Aufmerksamkeit!
Gibt es noch Fragen?

Zusätzliche Informationen

Overlog

- Erweiterung von Datalog durch:
  - Notation zur Spezifizierung des Speicherorts der Daten
  - SQL–basierte Erweiterung: Primärschlüssel, Aggregation
  - Modell zur Verarbeitung und Generierung von Änderungen an Tabellen
- Ereignisgesteuerte Programmiersprache
- Unterstützt relationale Tabellen

Zusätzliche Informationen

Overlog–Zeitschritt

- Overlog–Tupel werden in einem Zeitschritt verarbeitet
- Knoten sieht nur lokal gespeicherte Tupel (Tabelleneinträge)
- Kommunikation zwischen dem System und dem Knoten über Events
- Event: entspricht einem Einfügen, Updaten oder Löschen von Einträgen (Tupeln) in Datalog–Tabellen
- Änderungen erfolgen atomar und sind dauerhaft

Zusätzliche Informationen

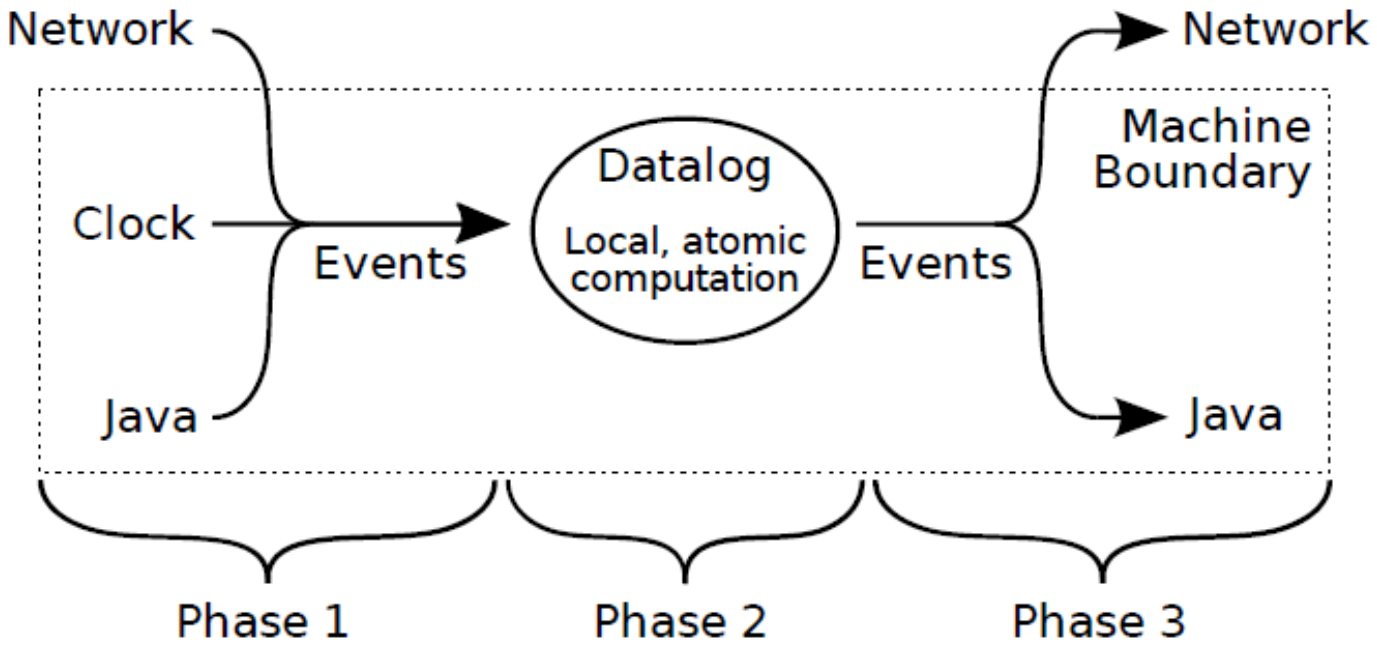


Abbildung: Quelle: BOOM: Data-Centric Programming in the Datacenter

Zusätzliche Informationen

Name	*Description*	*Relevant attributes*
job	Job definitions	jobid, priority, submit_time, status, jobConf
task	Task definitions	jobid, taskid, type, partition, status
taskAttempt	Task attempts	jobid, taskid, attemptid, progress, state, phase, tracker, input_loc, start, finish
taskTracker	TaskTracker definitions	name, hostname, state, map_count, reduce_count, max_map, max_reduce

Abbildung: BOOM–MR Relationen definieren Zustand des JobTrackers[5]

[5] Quelle: BOOM: Data-Centric Programming in the Datacenter

Zusätzliche Informationen

Name	*Description*	*Relevant attributes*
file	Files	fileid, parentfileid, name, isDir
fqpath	Fully-qualified pathnames	path, fileid
fchunk	Chunks per file	chunkid, fileid
datanode	DataNode heartbeats	nodeAddr, lastHeartbeatTime
hb_chunk	Chunk heartbeats	nodeAddr, chunkid, length

Abbildung: BOOM–FS Relationen definieren Metadaten des Dateisystems[6]

[6] Quelle: BOOM: Data-Centric Programming in the Datacenter