Verteilte Systeme

Map Reduce

Map Reduce

Problem:

Ein Rechen-Job (meist Datenanalyse/Data-Mining) soll auf einer riesigen Datenmenge ausgeführt werden. Teile der Aufgabe sind parallelisierbar, aber das Ergebnis muss sinnvoll zusammengefasst werden.

Infrastruktur:

- Großer Cluster von billigen Standardrechnern
- Standard Datennetz zwischen den Knoten
- Verteiltest Dateisystem, dass sich die Knoten teilen

Annahme:

Berechnungen lassen sich leichter verteilen als Daten

Map Reduce

Idee:

Den Job in zwei Funktionen/Phasen aufteilen, die sich gut verteilen und parallelisieren lassen:

Map: (k1, v2) -> list (k2, v2)

Abbildung von Key/Value Paaren auf andere Key/Value Paare, wobei die Keys k1, k2 nicht einzigartig sind!

Reduce: (k2, list(v2)) -> list (v2)

Für jeden Key (bzw Key range) aus dem Map-Schritt wird Reduce aufgerufen.

Ergebnis: Reduce(Map()) :: list(k2, list(v2))

Map Reduce: Beispiel

Problem:

Anzahl der Vorkommen von Worten in einem Text zählen

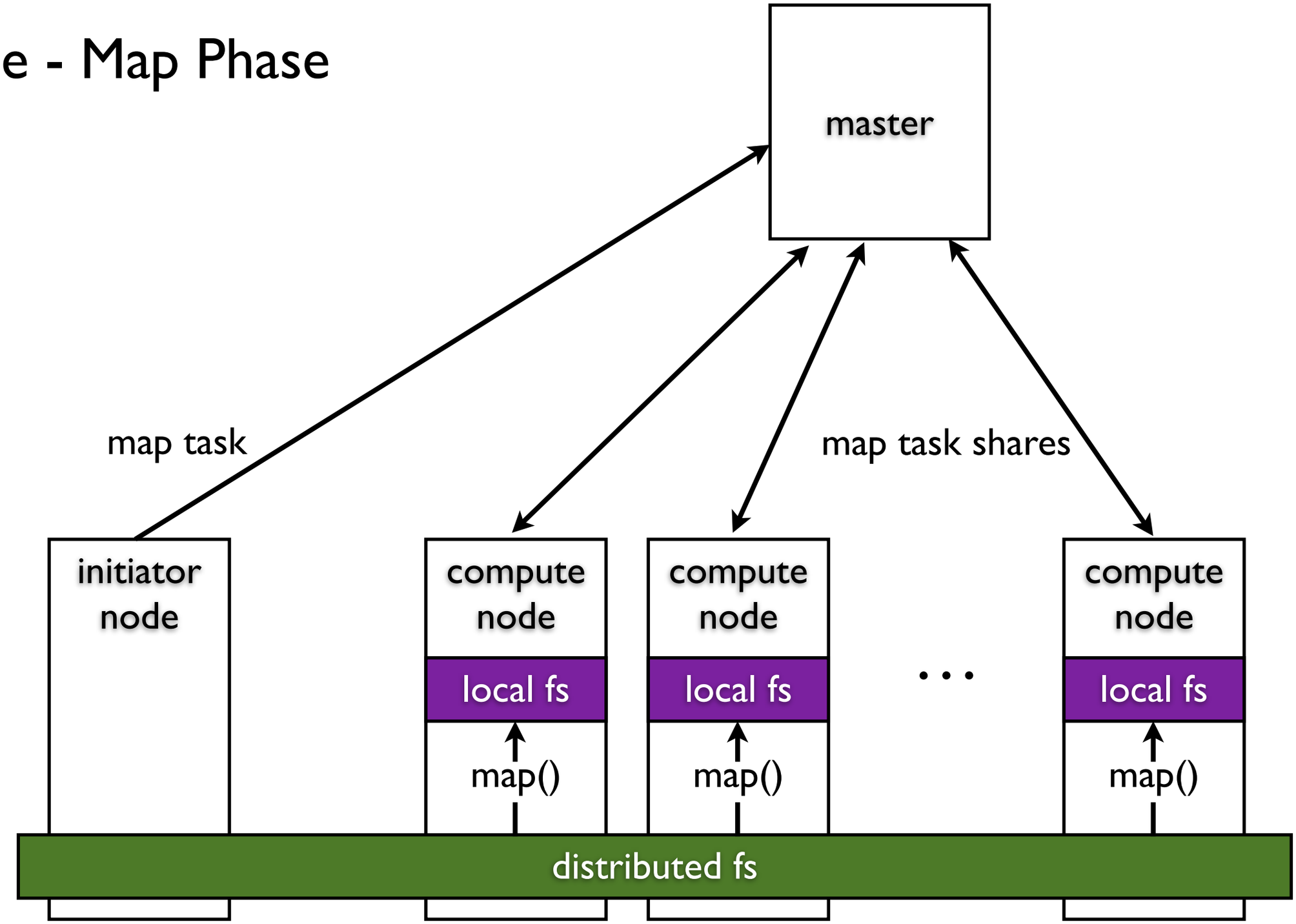
Split: Wir zerlengen den Text in Zeilen

Map: Für jedes Wort w_i in einer Zeile
emittieren wir: $(w_i, 1)$

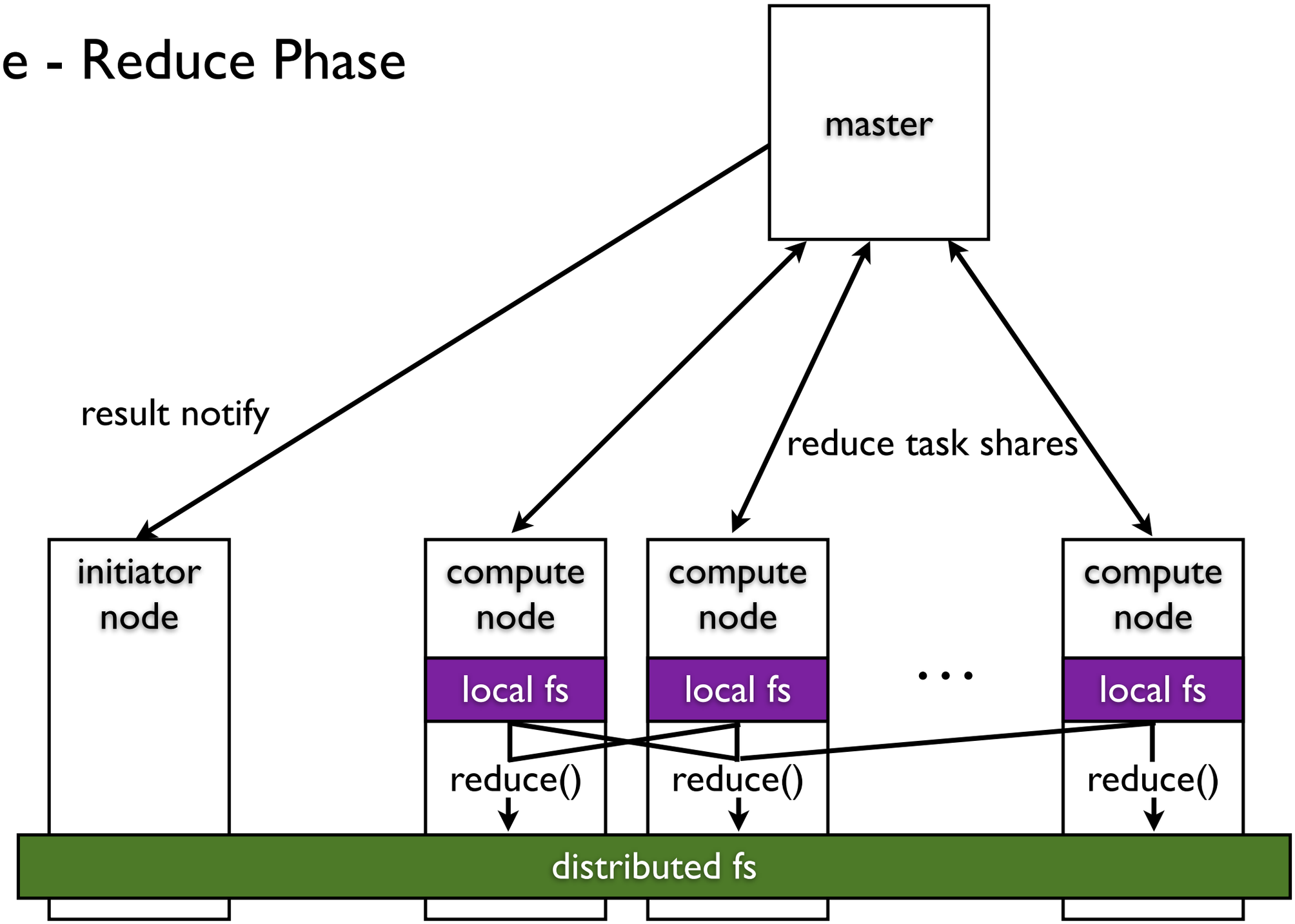
Reduce: Für alle Paare (w, 1) mit gleichem w
Emittiere die Anzahl.

Ergebnis: list(worte, anzahl)

Map Reduce - Map Phase



Map Reduce - Reduce Phase



Map Reduce: Google [Dean and Ghemawat 2004]

Architektur:

- Google Filesystem (GFS) als Verteiltes Dateisystem
- Standardhardware als Compute-Knoten, über mehrere Rechenzentren verteilt

Ablauf:

1. Initiator Teilt die Eingabedaten in Teile von 16-64MB und startet den Master- und die Compute-Knoten
2. Master Teilt den freien Compute-Knoten Map-Jobs zu
3. Compute-Knoten führen Map-Jobs aus und melden Vollzug, Ergebnisse werden auf lokal geschrieben.
4. Master startet nach Erfolg der Map-Jobs Reduce-Jobs
4. Compute-Knoten führen Reduce-Jobs aus und melden Vollzug, Ergebnisse werden ins GFS geschrieben.

Map Reduce: Google [Dean and Ghemawat 2004]

Aufteilen der Eingabedaten:

- Die Anwendungen, die Eingabedaten produzieren, legen diese in Chunks (64 MB) ab, so dass Datensätze nie über Chunkgrenzen gehen.
- Der Master vergibt die Map-Jobs auf Chunks oder Teilen von Chunks (typischerweise 16-64 MB)
- Der Map-Reduce-Master fragt den GFS-Master an, auf welchen Knoten die Chunks (bzw Replikate der Chunks) liegen und versucht auf diesen Knoten oder auf möglichst nahen Knoten die Map-Jobs auszuführen.

Map Reduce: Google [Dean and Ghemawat 2004]

Ablage der Zwischenergebnisse:

- Die Zwischenergebnisse werden mittels einer anwendungs-spezifischen Hash-Funktion nach Key in Buckets sortiert,jeder Bucket entspricht einer Datei, in der Daten aller Keys mit gleichem Hash unsortiert liegen.
- Zwischenergebnisse werden so lange wie möglich im RAM gepuffert und in großen Blöcken lokal geschrieben.
- Der Map-Job meldet dem Master die Hashwerte und die dazugehörigen Dateinamen zurück.

Map Reduce: Google [Dean and Ghemawat 2004]

Ausführen der Reduce-Funktion:

- Für jeden Bucket wird ein Reduce-Job gestartet
- Reduce-Jobs trennen ggf. die Buckets nach Keys k2 auf
- Für jeden Key wird reduce(k2, liste(v2)) aufgerufen.
- Die Ausgabedaten werden dann, ggf. nach keys sortiert, in die Ausgabedatei(en) ins GFS geschrieben.

Map Reduce: Google [Dean and Ghemawat 2004]

Ausfall von Compute-Knoten:
Da sehr viele Standardrechner beteiligt sind, sind Ausfälle normal und müssen toleriert werden.

- Der Master bekommt regelmäßig Bestätigungen von allen aktiven Jobs.
  Beliben diese aus, wird der Job auf einem anderen Knoten neu gestartet.
- Für einzelne Jobs, die zu lange laufen, werden ggf. zusätzliche Instanzen gestartet, die das Ergebnis der ersten instanz, die Terminiert, wird übernommen.
- Nicht verfügbare Zwischenergebnisse werden ggf. durch Wiederholen der Jobs erneut berechnet.

Map Reduce: Google [Dean and Ghemawat 2004]

Abbruch von Jobs:

Durch Fehler in den Eingabedaten kann es passieren, dass die Prozesse für Map oder Reduce Jobs abstürzen:

- Jeder Prozess schreibt die Position des aktuell bearbeiteten Datensatzes in eine Flag-Datei.
- Beim Absturz (z.B. Segmentation Fault) wertet ein Wrapper-Programm das Flag-File aus und meldet den Absturz an den Master
- Falls ein Datensatz mehrfach Abstürze erzeugt, wird dieser vom Master übersprungen.

Verteilte Systeme

Google File System (GFS)

GFS [Ghemawat, Gobioff and Leung 2003]

Problemstellung:

Terabytes an Daten sollen auf günstiger Standardhardware hochverfügbar und schnell verarbeitet und gespeichert werden

Annahmen:

- Wenige Dateien mit Größen zwischen 100MB und 100GB
- Praktisch nur sequentielles schreiben in großen Blöcken, verteiltes Schreiben darf langsam sein
- Gemischt sequentielles lesen in großen Blöcken und verteiltes lesen in kleinen Blöcken.
- Konkurrierendes Schreiben ist nur anhängend
- Hohe Bandbreite ist wichtiger als zuverlässige Latenz

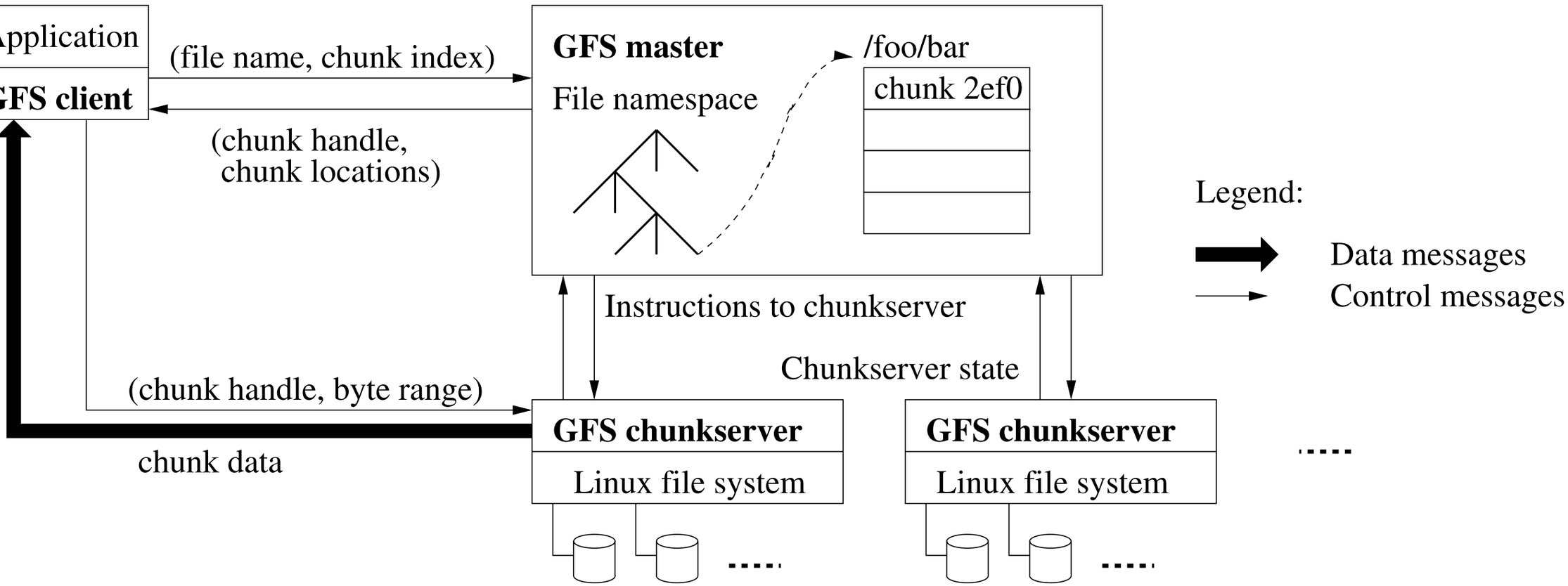
GFS [Ghemawat, Gobioff and Leung 2003]

Architektur:

- Einzelner “GFS-Master” verwaltet die Meta-Daten
  - einfach zu implementieren
  - wenig overhead
- Klienten lesen nur Meta-Daten vom Master und cachen sie.
- Master-Server sendet alle Meta-Daten-Änderungen an einen Slave, der bei Ausfall die Aufgabe des Masters übernehmen kann.
- Nutzdaten “Chunks” liegen auf vielen Servern verteilt, Klienten lesen diese direkt von den Chunk-Servern.
- Jede Datei hat einen Replikationsfaktor, wieviele Replikate ihrer Chunks existieren sollen.
- Chunks sind 64MB groß.

GFS [Ghemawat, Gobioff and Leung 2003]

Architektur:



GFS [Ghemawat, Gobioff and Leung 2003]

Dateioperationen und Konsistenzgarantien

GFS verwendet für alle Dateioperationen deutlich abgeschwächte Konsistenzgarantien, als bei den meisten Dateisystemen üblich, um den Datendurchsatz steigern zu können:

- read (file, offset, size)
  liefert Daten an einem bestimmten Offset der Datei.

GFS [Ghemawat, Gobioff and Leung 2003]

Dateioperationen und Konsistenzgarantien

- write (file, offset, data)
  ersetzt die Daten an offset.

  Falls write erfolgreich war, garantiert GFS, dass alle lebendigen Kopien die gleichen Daten enthalten.

  Für konkurrierende Schreiboperationen wird keine weitere Konsistenz gewährt: Teile der ersetzten Region können durch andere, konkurrierende Schreiboperationen nichtsequentialisierbar überschrieben werden!)

GFS [Ghemawat, Gobioff and Leung 2003]

Dateioperationen und Konsistenzgarantien

- recordAppend (file, data)
  Fügt einen Datensatz an eine Datei an.
  GFS wählt die Position selbständig.

Falls die Operation erfolgreich war, würde das Datum mindestens einmal in allen Kopien geschrieben.

GFS [Ghemawat, Gobioff and Leung 2003]

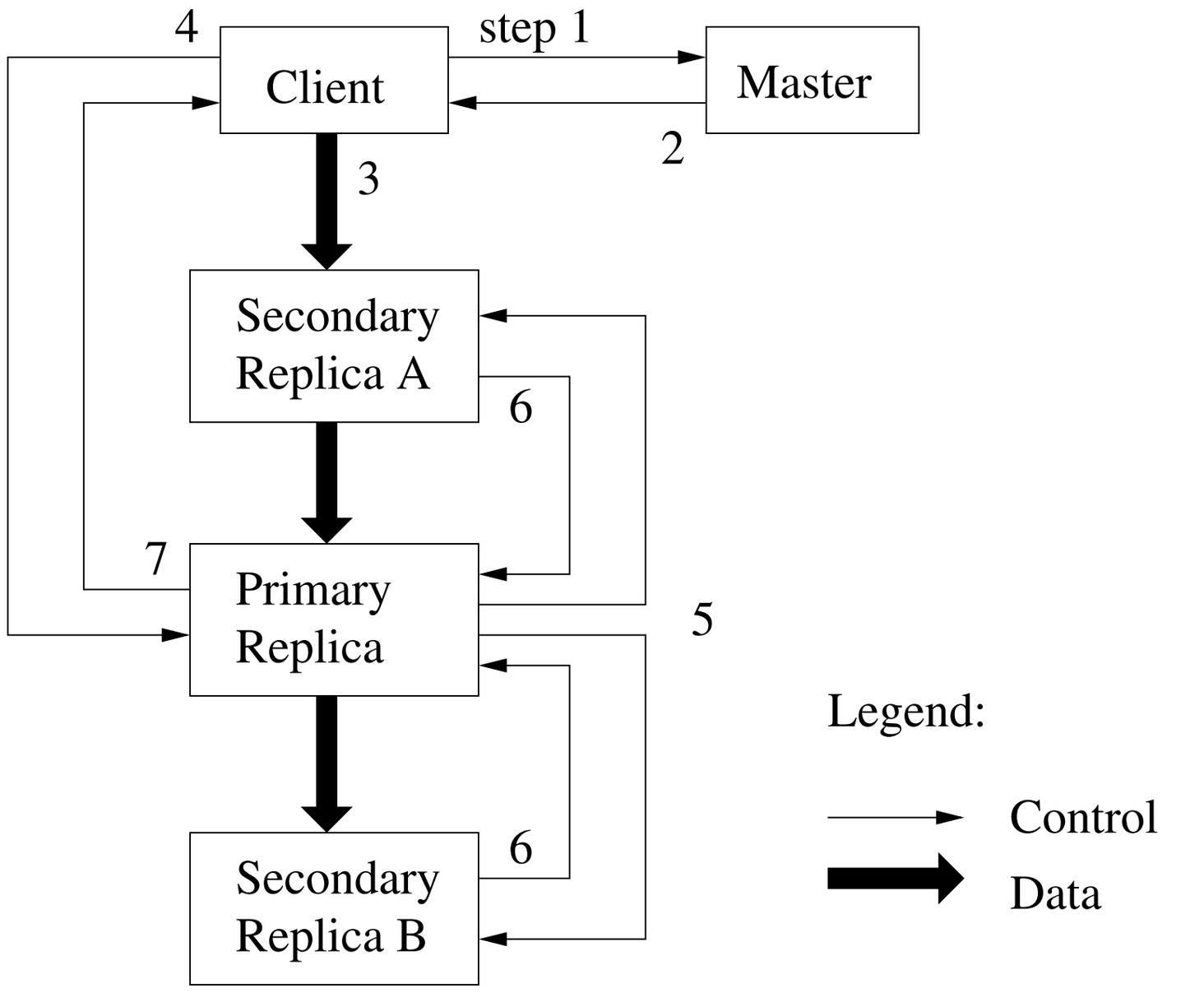
Dateioperationen und Konsistenzgarantien (Fazit)

Anwendungen müssen sich auf die schwächeren Garantien einstellen.

Die meisten Anwendungen bei Google schreiben Daten nur durch anhängen und enthalten Code um duplizierte Datensätze zu erkennen. Der unnötig belegte Speicherplatz wird zu Gunsten des höheren Datendurchsatzes in Kauf genommen.

GFS [Ghemawat, Gobioff and Leung 2003]

Commit-Protokoll:



GFS [Ghemawat, Gobioff and Leung 2003]

Metadatenoperationen

Alle Metadaten-Operationen werden atomar durch den GFS-Master realisiert, dabei wird versucht die Datenmenge zu minimieren und alle Daten im Arbeitsspeicher zu halten.

Operationen beinhalten das Anfordern von Leases auf Chunks, Erzeugen und Löschen von Dateien, Exklusiver Zufriff auf Dateien und das Anfertigen von Snapshots.

GFS [Ghemawat, Gobioff and Leung 2003]

Sperren Namensraum

GFS verwendet keine Verzeichnisse im herkömmlichen Sinne, kann aber Sperren auf Teilen des Namensraumes (z.B. für Snapshots) durchsetzten.

Beispiel: Zum Erzeugen der Datei “/home/foo/file” wird eine Lesesperre für “/home” und “/home/foo” und eine Schreibsperre für “/home/foo/file” angefordert.

GFS [Ghemawat, Gobioff and Leung 2003]

Snapshots

GFS bietet atomare Kopien Snapshots für Unterbäume des Namensraumes, die dann in einem anderen Unterbaum des verfügbar sind.

Realisierung:

1. Alle Chunk-Leases werden invalidiert
2. Der Namensraum wird auf dem GFS-Master kopiert, die Chunks entsprechend markiert
3. Bei späteren Anforderung von Leases für Schreib-zugriffe werden die Chunks erst kopiert und dann Leases für die Kopien erteilt.

Verteilte Systeme

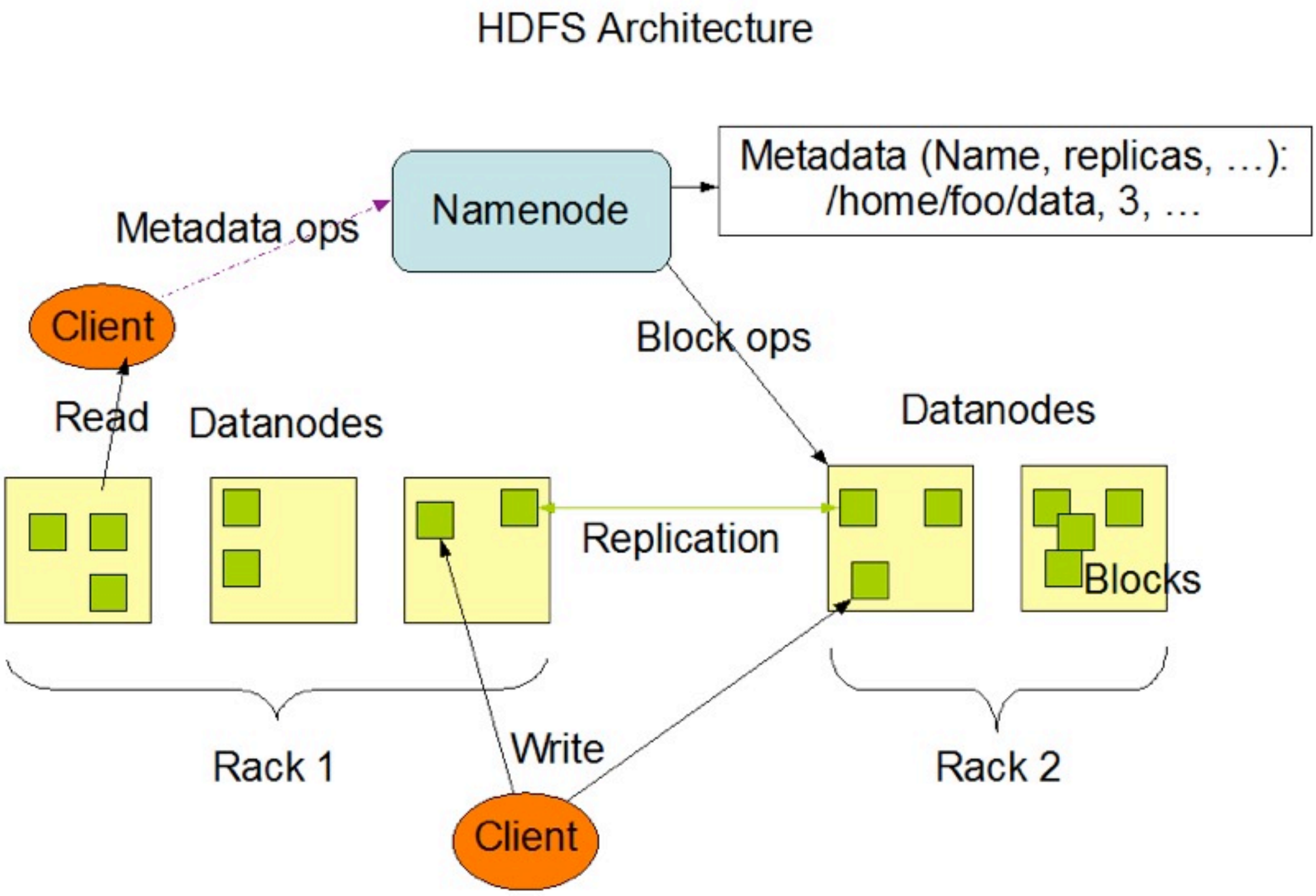
Hadoop

Hadoop

Projekt der Apache-Fundation, das eine freie Map/Reduce Infrastruktur basierend auf Java anbietet.

- Stark angelegt an Googles Map/Reduce
- Operationen als Java-Klassen
- HDFS als Hintergrundspeicher (statt GFS)
- Viele Erweiterungen (Domänenspezifische Sprachen, für bestimmte Szenarien optimierte Datenbanken)

HDFS


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Hadoop

```
public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
          String line = (caseSensitive) ? value.toString() : value.toString
().toLowerCase();

          for (String pattern : patternsToSkip) {
            line = line.replaceAll(pattern, "");
          }

          StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
            reporter.incrCounter(Counters.INPUT_WORDS, 1);
          }
    }
```

Hadoop

```
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
    int sum = 0;
        while (values.hasNext()) {
        sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
```