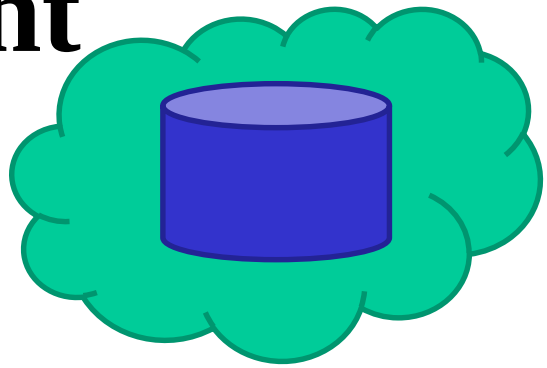
Cloud Data Management



Kapitel 5: MapReduce und Datenbanken

Dr. Anika Groß

Wintersemester 2016

Universität Leipzig
Institut für Informatik
http://dbs.uni-leipzig.de

Inhaltsverzeichnis

SQL-Anfrageformulierung mit MapReduce

- (manuelle) Umschreibung SQL $\rightarrow$ MapReduce

- Beispiel: CouchDB
  - Dokumenten-orientierte Datenbank
  - kein Schema
  - Dokumente in JSON-Format
- Anfragen durch View-Definitionen
  - Definition von map- und reduce-Funktion in Javascript (und anderen Sprachen)

Beispieldaten

- Konzeptionell: Repräsentation als verschachtelte Tabelle

id	name	time	user	camera	info			tags
					width	height	size	
1	fish.jpg	17:46	bob	nikon	100	200	12345	[tuna, shark]
2	trees.jpg	17:57	john	canon	30	250	32091	[oak]
3	snow.png	17:56	john	canon	64	64	1253	[tahoe, powder]
4	hawaii.png	17:59	john	nikon	128	64	92834	[maui, tuna]
5	hawaii.gif	17:58	bob	canon	320	128	49287	[maui]
6	island.gif	17:43	zztop	nikon	640	480	50398	[maui]

- Intern: Repräsentation als Dokumentenmenge (JSON-Format)

```
{“_id”:”1”, "name":"fish.jpg",”time”:”17:46","user":"bob“,"camera":"nikon",
            "info":{"width":100,"height":200,"size":12345},"tags":["tuna","shark"]}
{“_id”:”2”, "name":"trees.jpg",“time”:”17:57”,"user":"john”,"camera":"canon",
            "info":{"width":30,"height":250,"size":32091},"tags":["oak"]}
....
```

Quelle: http://labs.mudynamics.com/wp-content/uploads/2009/04/icouch.html

Selektion

- Selektion = Bedingung für Attributwert(e)
  - SQL: ... WHERE attr = “xy”
- MapReduce
  - map: Prüfung durch IF-Bedingung, Ausgabe der selektierten Dokumente
  - reduce: Id-Funktion
- Beispiel
  - SQL: SELECT * FROM table WHERE user = “bob”

id	name	time	user	camera	info			tags
					width	height	size	
1	fish.jpg	17:46	bob	nikon	100	200	12345	[tuna, shark]
5	hawaii.gif	17:58	bob	canon	320	128	49287	[maui]

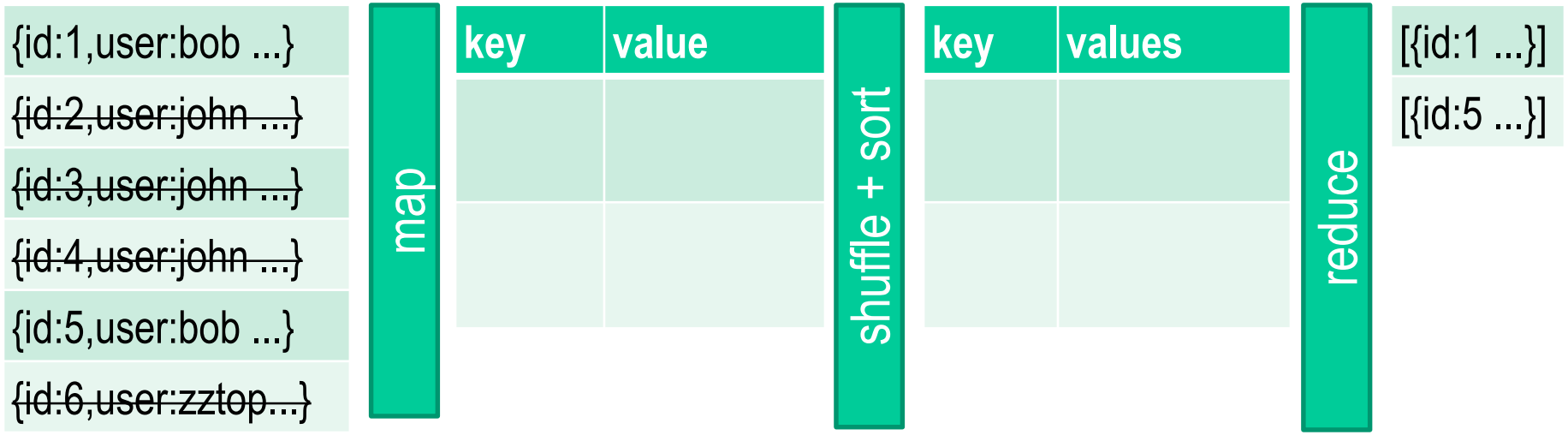
Selektion: Beispiel

map

```
function (doc) {
  if (doc.user == "bob")
    emit (doc.id, doc);
}
```

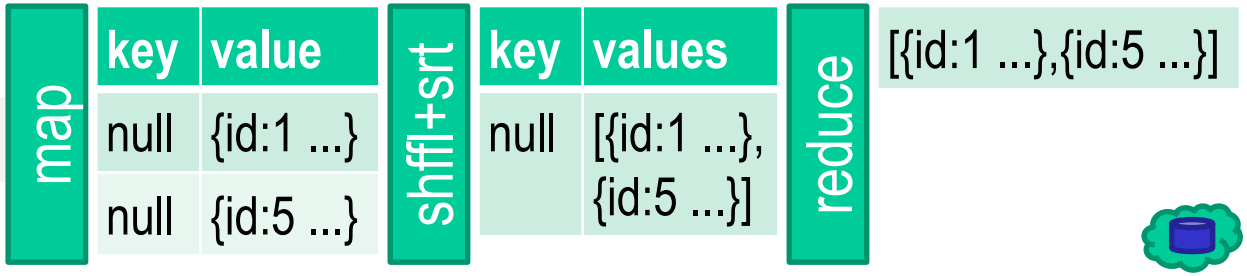
reduce

```
function (key, values) {
  return values;
}
```



Alternative

```
emit (null, doc);
```



Projektion

- Projektion = Einschränkung des Ergebnisses auf Attribute
  - SQL: SELECT Attr1, Attr2 FROM ...
- MapReduce
  - map: Generierung eines neuen Dokuments mit entsprechenden Attributen
  - reduce: Id-Funktion
- Duplikateliminierung
  - map: Key = Attribut(kombination)
  - reduce: Ausgabe des ersten Values
- Beispiel
  - SQL: SELECT (DISTINCT) user FROM table

user
bob
john
john
john
bob
zztop

user
bob
john
zztop

Projektion: Beispiel (ohne Duplikateliminierung)

```
map
function (doc) {
  emit(doc.id,{"user":doc.user});
}
```

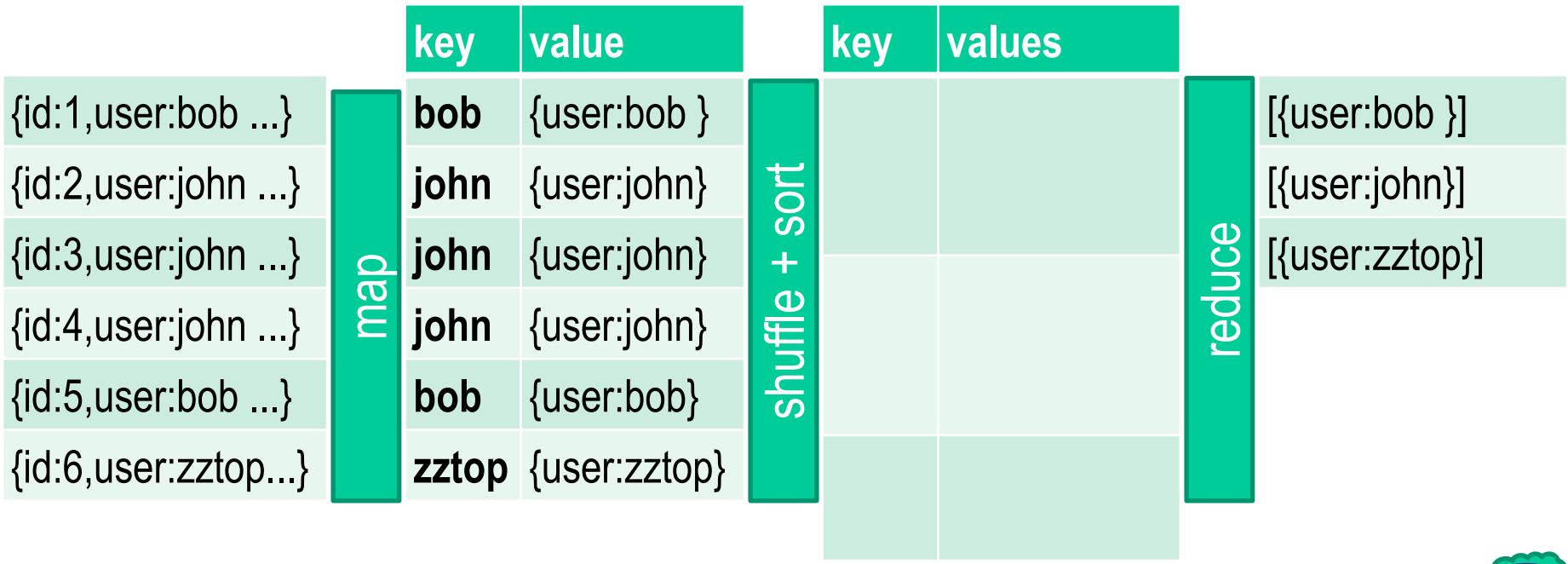
```
reduce
function (key, values) {
  return values;
}
```

		key	value		key	values		
{id:1,user:bob ...}	map	1	{user:bob }	shuffle + sort	1	[{user:bob }]	reduce	[{user:bob }]
{id:2,user:john ...}		2	{user:john}		2	[{user:john}]		[{user:john}]
{id:3,user:john ...}		3	{user:john}		3	[{user:john}]		[{user:john}]
{id:4,user:john ...}		4	{user:john}		4	[{user:john}]		[{user:john}]
{id:5,user:bob ...}		5	{user:bob}		5	[{user:bob}]		[{user:bob}]
{id:6,user:zztop...}		6	{user:zztop}		6	[{user:zztop}]		[{user:zztop}]

Projektion: Beispiel (mit Duplikateliminierung)

```
map
function (doc) {
  emit(doc.user,{"user":doc.user});
}
```

```
reduce
function (key, values) {
  return values[0];
}
```



Gruppierung und Aggregatfunktion

- Gruppierung
  - Zusammenfassen von Datensätzen mit gleichen Attributwerten
  - Bildung aggregierter Attributwerte pro Gruppe durch Aggregatfunktionen (z.B. SUM)
- MapReduce
  - map: Gruppierungsattribut(e) als Key
  - reduce: Anwendung der Aggregatfunktion
- Beispel
  - SELECT camera, AVG(info.size) as avgsize
    FROM Table
    GROUP BY camera

camera	avgsize
canon	27543.7
nikon	51859

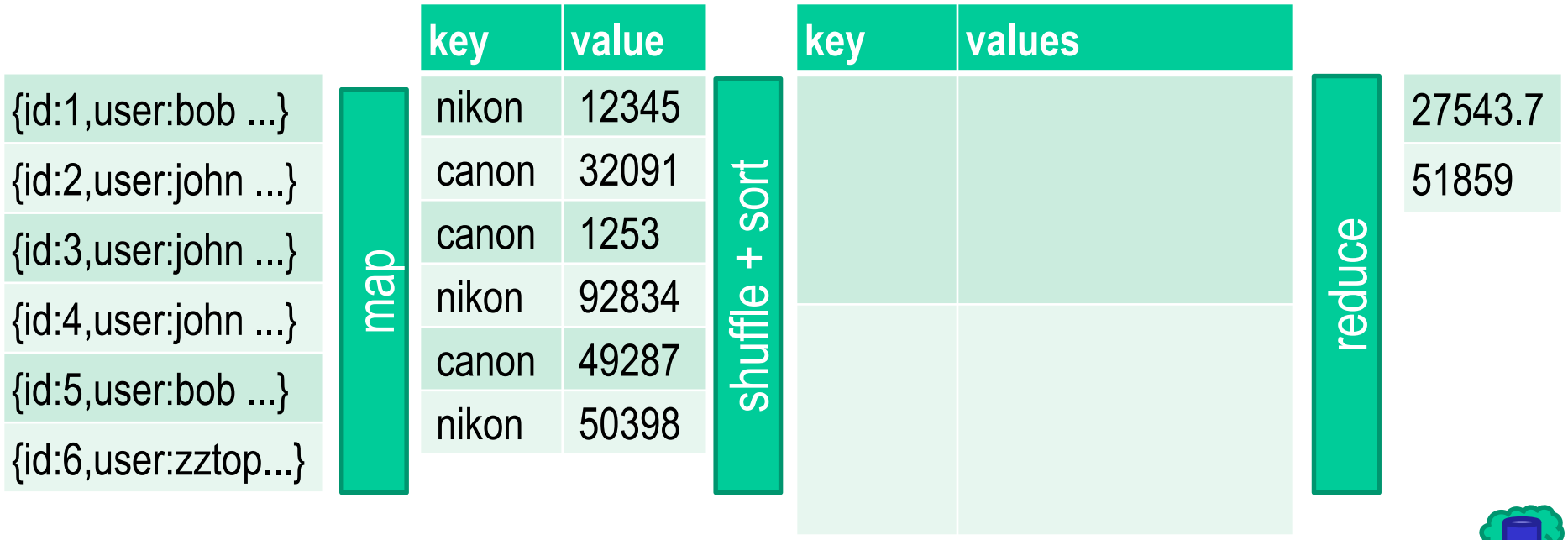
Gruppierung und Aggregatfunktion: Beispiel

map

```
function (doc) {
  emit(doc.camera,
   doc.info.size);
}
```

reduce

```
function (key, values) {
  sum = 0;
  for (i=0; i<values.length; i++) {
    sum = sum + values[i];
  }
  return sum/values.length;
}
```



Equi-Join + Mehrwertiges Attribut

- Equi-Join = Verknüpfung zweier Relationen über Attributgleichheit
  - SQL: ... WHERE Tab1.Attr1 = Tab2.Attr2
- Mehrwertige Attribute in 1NF
  - 1:N/N:M-Beziehung = weitere Relation(en), die durch Join verknüpft werden
- MapReduce
  - map: Join-Attribut als Key
  - reduce: Iteration über Paare
- Beispiel (SQL)
  - Welche Bilder haben (mind.) ein gemeinsames Tag

id	name	time	user	camera	info			tags
					width	height	size	
1	fish.jpg	17:46	bob	nikon	100	200	12345	[tuna, shark]
2	trees.jpg	17:57	john	canon	30	250	32091	[oak]
3	snow.png	17:56	john	canon	64	64	1253	[tahoe, powder]
4	hawaii.png	17:59	john	nikon	128	64	92834	[maui, tuna]
5	hawaii.gif	17:58	bob	canon	320	128	49287	[maui]
6	island.gif	17:43	zztop	nikon	640	480	50398	[maui]

name1	name2
fish.jpg	hawaii.png
hawaii.png	island.gif
hawaii.gif	hawaii.png
hawaii.gif	island.gif

Equi-Join + Mehrwertiges Attribut: Beispiel

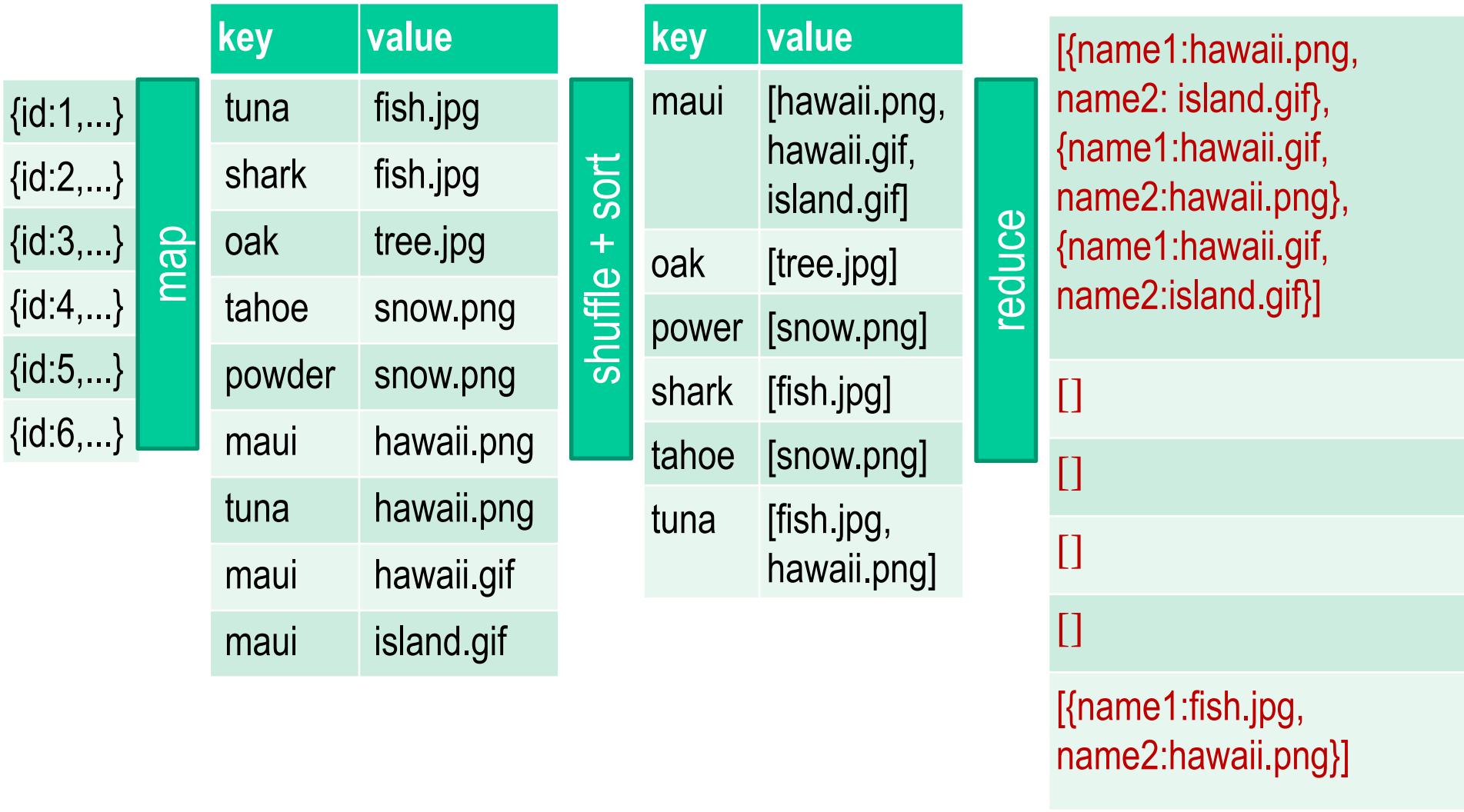
map

```
function (doc) {
  for (i=0; i<doc.tags.length; i++) {
  emit (doc.tags[i], doc.name);
}
```

reduce

```
function (key, values) {
  var result = new Array();
  for (i=0; i<values.length; i++) {
    for (k=0; k<values.length; k++) {
      if (i<k) {
        result.push ({“name1”:values[i],
                      “name2”:values[k]});
      }
    }
  }
  return result;
}
```

Equi-Join + Mehrwertiges Attribut: Beispiel (2)



Inhaltsverzeichnis

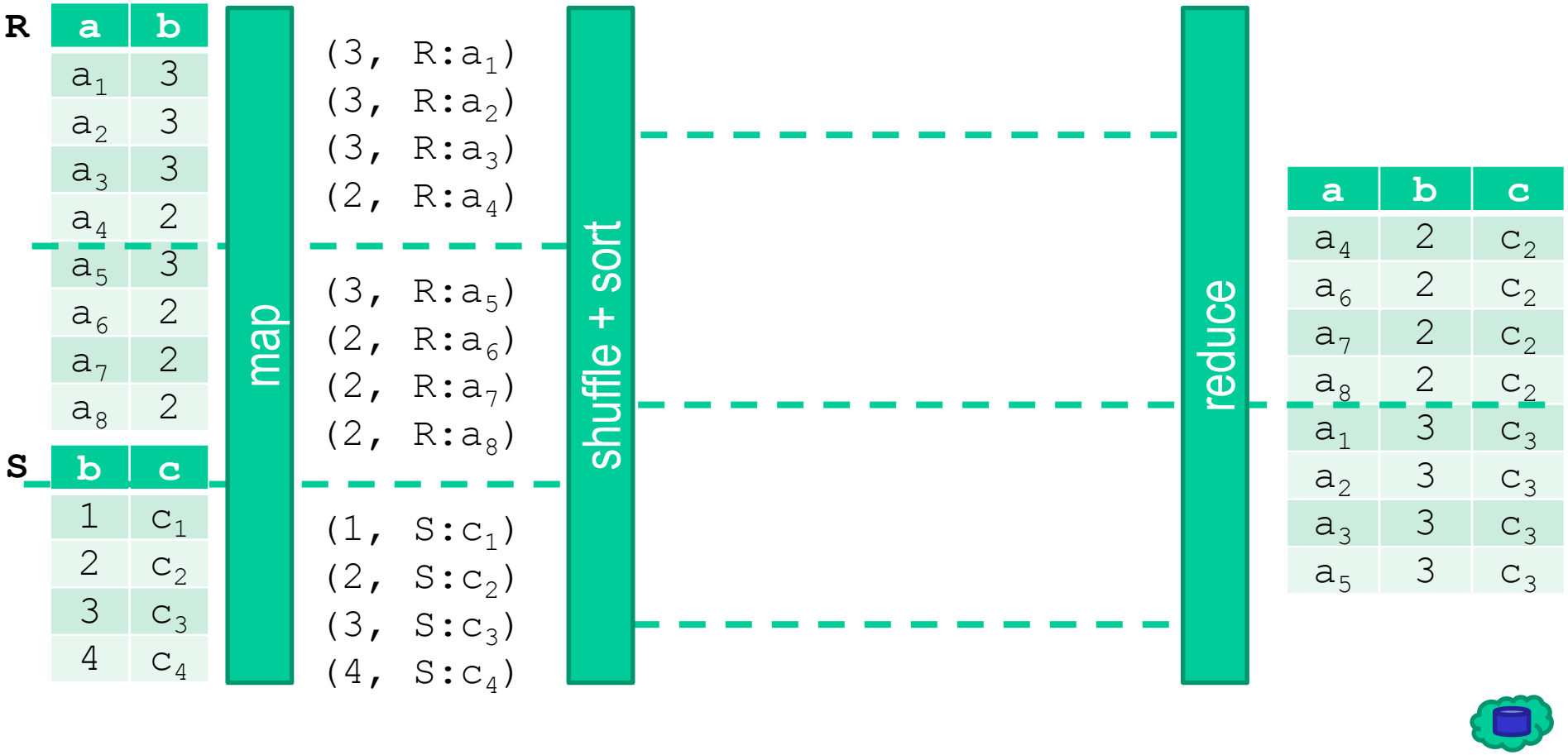
Join-Realisierung mit MapReduce

- Join ist wichtige und teure Datenbank-Operation
  - unterschiedliche Join-Arten (Natural, Outer, ...), Anzahl beteiligter Relationen
  - Fokus im Folgenden: Natural Join zwischen R und S
- Häufiger Anwendungsfall für WebApps: Logfile-Auswertung
  - Logfile ⋈ User über Attribut UserId
  - Logfile (#Klicks) meist deutlich größer als Referenztabelle (#User)
  - Geringe Join-Selektivität (nur x% der User pro Tag auf Website)
- Join-Performanz u.a. abhängig von
  - gleichmäßiger Lastbalancierung der Knoten (z.B. Lastbalancierung Entity Matching)
  - Datenmenge, die zwischen Map- und Reduce-Phase sortiert und transferiert wird (ggf. unter Berücksichtigung der Lokalität)
- Verschiedene Verfahren
  - Repartition Join
  - Broadcast Join
  - Semi-Join

Quelle diese und nächste Folien: [MRJoin]

Repartition Join

- Naiver Ansatz
  - map: Key=Join-Attribut, Value=Relationsname + Nicht-Join-Attribute
  - reduce: Alle Paare mit unterschiedlichen Relationsnamen



Repartition Join: Nachteile

- Alle Daten werden zwischen Map- und Reduce-Phase sortiert und an die Reduce Tasks geschickt
  - Verbesserung durch Broadcast Join, Semi-Join (nächste Folien)
- Reducer muss alle N Datensätze pro Key puffern
  - keine Reihenfolge bzgl. Input-Relation, da nur nach Join-Attribut sortiert
  - Hadoop-Implementierung erlaubt nur sequentiellen Datenzugrifff
- Lösung
  - Erweiterung des Map-Keys **und** des Values um Relationsname
  - Sortierung: Keys der kleineren Relation (*S=0*) vor Keys der größeren Relation (*R=1*)
  - Reduce muss nur noch Datensätze von *S* puffern (erkannt am Wechsel d. 2. Komponente des Values)
- Beispiel

normal

(2, R:a_4)
(2, S:c_2)
(2, R:a_6)
(2, R:a_7)
(2, R:a_8)

erweiterter Key+Sortierung

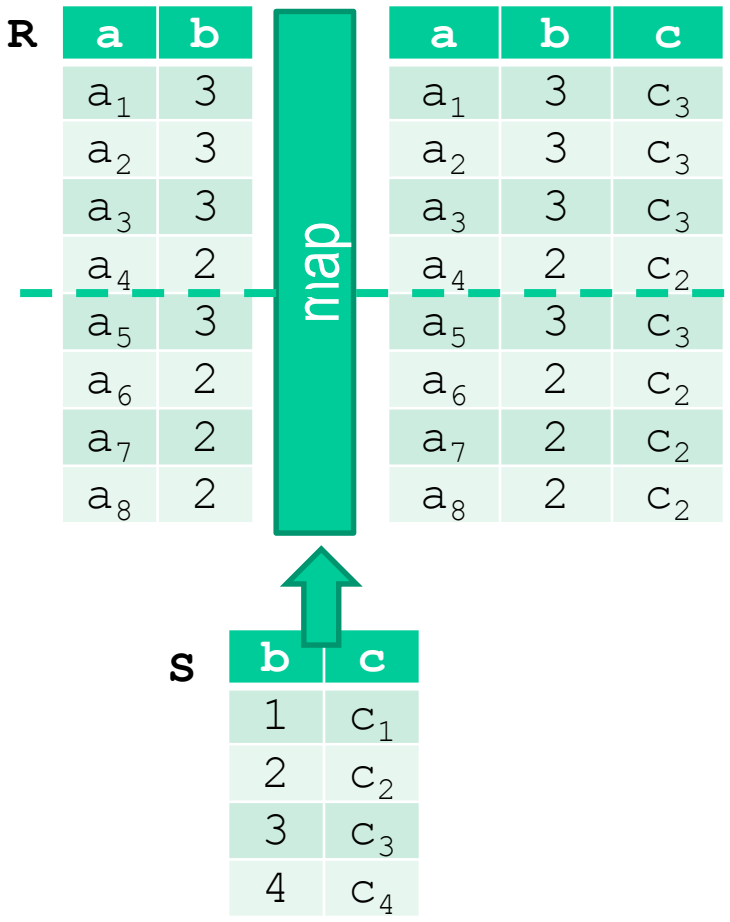
(2:0, c_2:0)
(2:1, a_4:1)
(2:1, a_6:1)
(2:1, a_7:1)
(2:1, a_8:1)

Nach Gruppierung

(2, [c_2:0,
a_4:1,
a_6:1,
a_7:1,
a_8:1])

Broadcast-Join

- Idee
  - Kleinere Relation (*S*) als zusätzlichen map Input
  - Join nur in map-Phase, kein Reduce notwendig
- Notwendiger Datentransfer
  - kleinere Relation muss an alle *n* Knoten geschickt werden → $n \cdot |S|$ Datensätze
  - kein Transfer von R, da jeder map-Task seine "lokale" map-Partition bearbeitet
- Vergleich Repartition-Join
  - beide Relationen werden auf alle reduce-Tasks aufgeteilt → $|R|+|S|$ Datensätze



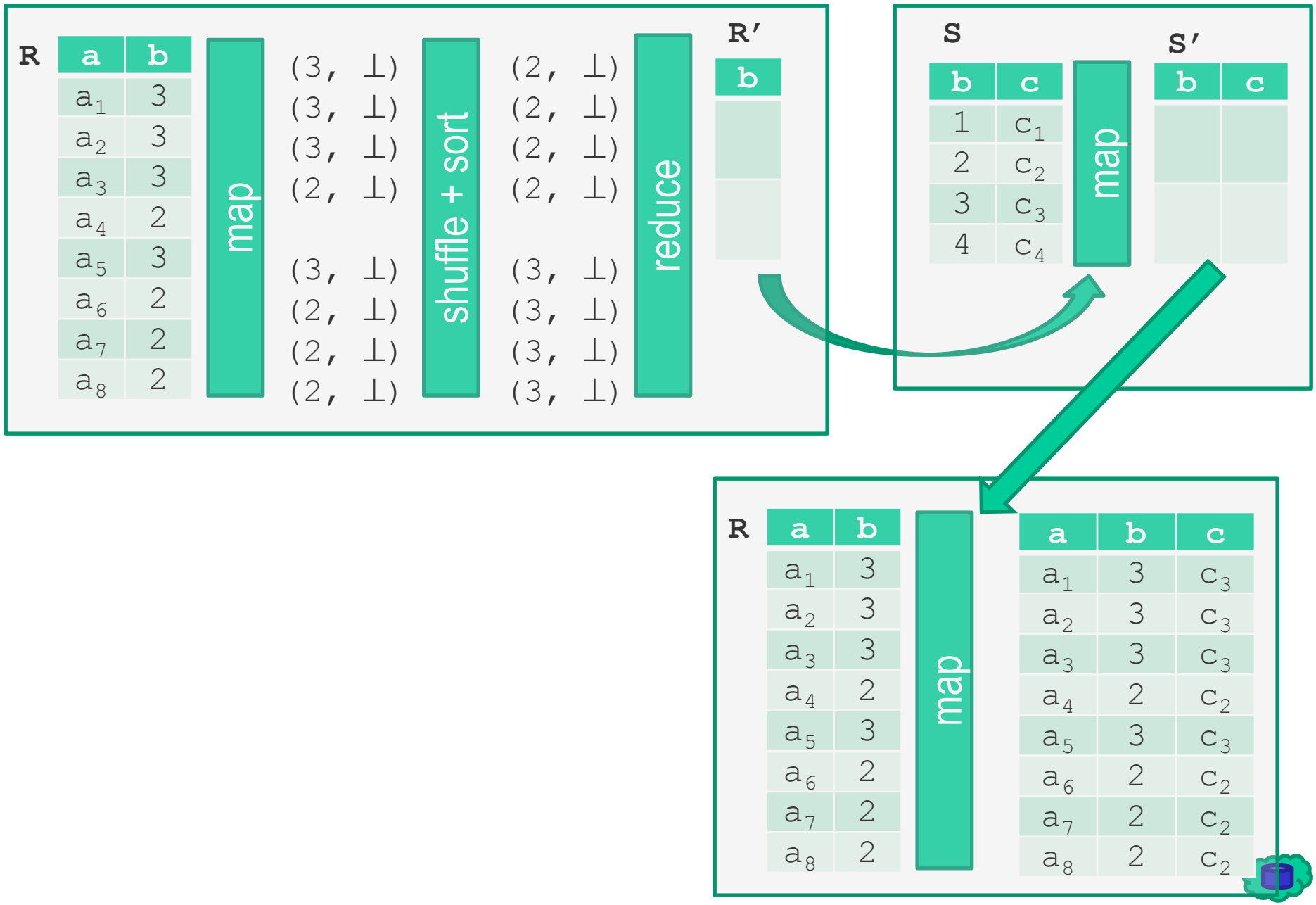
- Dynamische Entscheidung möglich, welche Relation kleiner und ob Broadcast-Join vorteilhaft ist

Semi-Join

- Neben Anzahl der Datensätze auch Größe der zu transferierenden Datensätze entscheidend
  - Größe (Join-Attribut) << Größe (Datensatz)
- Semi-Join-Idee: $R \bowtie S = R \bowtie (\Pi_b (R) \bowtie S)$ mit Join-Attribut *b*
  - Realisierung durch drei Map-Reduce-Schritte

1. $R' = \Pi_b (R)$
   - map extrahiert Join-Attribut, 1 reduce task entfernt Duplikate
2. $S' = R' \bowtie S$
   - Broadcast-Join mit "kleinerer Relation" R'
3. $R \bowtie S'$:
   - Broadcast-Join mit kleinerer Relation S'

Semi-Join: Beispiel



Evaluation in [MRJoin]

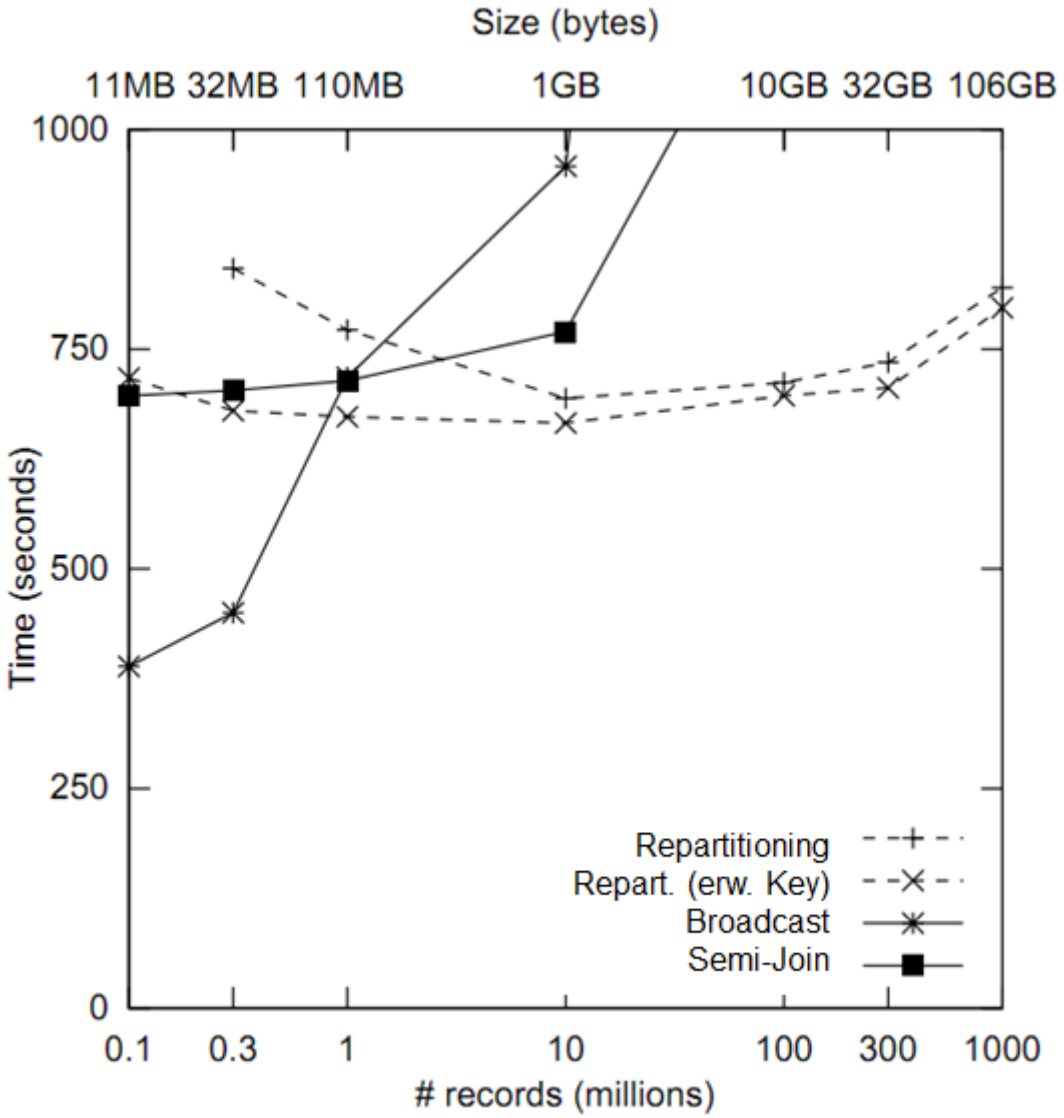


Tabelle: Datenmenge, die durch das Netzwerk geschickt wird

- geringer als Map-Output (z.T. gleicher Knoten für map und reduce-Task)

#Datensätze (Relation S)	Repartition (erw. Key)	Broad-cast
0.3 Millionen	145 GB	6 GB
10 Millionen	145 GB	195 GB
300 Millionen	151 GB	6240 GB

- Broadcast für kleine S
- Repartitioning: Nutzen des erweiterten Keys
- Semi-Join muss zweimal (große) Relation R einlesen

Inhaltsverzeichnis

- SQL-Anfrageformulierung mit MapReduce
- Joins mit MapReduce
- **Data Warehousing / Datenanalyse mit MapReduce**
  - Hive
  - Pig

Vergleich: Hadoop/MR vs. Parallele DBS

	Hadoop / MapReduce	Shared Nothing-RDBMS
Datengröße	PB	TB-PB
Struktur	Semistrukturierte Daten	Statisches Schema
Partitionierung	Blöcke in DFS (Byteweise)	Horizontal
Anfrage	MapReduce-Programme	Deklarativ (SQL)
Zugriff	Batch	Punkt/Bereich via Indexes
Updates	Write once read many times	Read and write many times
Scheduling		
Verarbeitung		
Datenfluss		
Fehlertoleranz		
Skalierbarkeit	Linear, unbegrenzt	Linear (existierende Setups), begrenzt
HW-Umgebung	Heterogen (preiswerte Standard-HW)	Homogen (teure High-End-Hardware)
SW-Kosten	Frei / Open Source	Sehr teuer

Vorteile: MapReduce vs. Parallele DBS

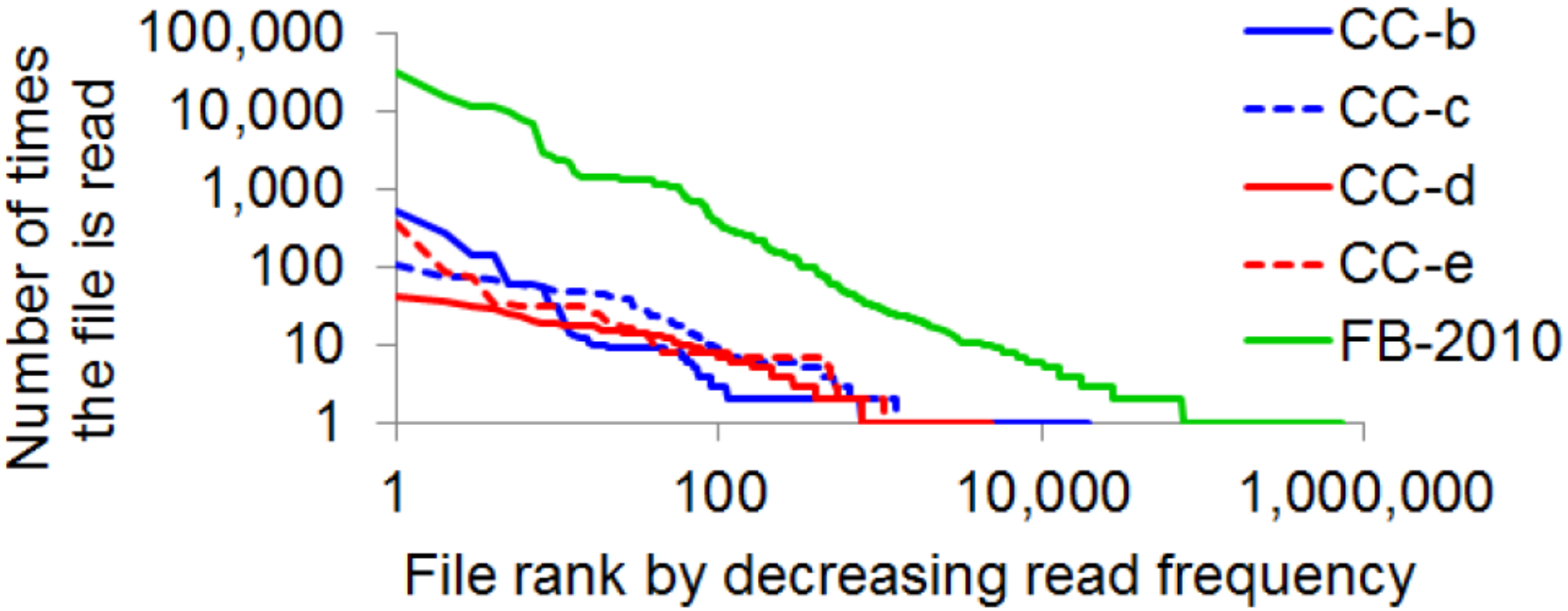
- Vorteile MapReduce
  - Skalierbarkeit/Fehlertoleranz
  - Konfigurationsaufwand
  - Kosten
  - Kein initialer Ladevorgang
- Vorteile SN-DBS
  - Deklarative Anfragesprache
  - Anfragen werden um Größenordnungen schneller beantwortet
  - (Zzt.) Arbeit auf komprimierten Daten
  - Random Access
- Typische Anwendungsfälle MapReduce
  - ETL
  - Data-Mining, Data-Clustering
  - Analyse semistrukturierter Daten (Web-Logs, …)
  - Einmal-Analysen eines Datenbestandes

Datenanalyse: Beispiel Facebook

- Facebook
  - 4TB komprimierte Daten pro Tag
  - 135TB komprimierte Daten werden pro Tag analysiert
- Aggregationen
  - Anzahl Clicks/Pageviews pro Tag/Monat/...
- Ad-hoc-Analyse
  - Wieviele Foto-Uploads zu Neujahr in den USA pro County/State?
- Data Mining
  - Nutzerverhalten als Funktion von Attributen (#Pageviews, #Sessions, Zeit, ...)
- Spam-Erkennung
  - (Verdächtige) häufige Muster in UGC (user generated content)
- Auswertung / Optimierung von Werbung
  - Anzahl AdClicks pro Nutzertyp/...

Datenanalyse: Access Skew

- Studie mehrerer Unternehmen (u.a. Facebook) zeigt Access Skew
  - Wenige Dateien werden sehr häufig gelesen
  - Viele Dateien sehr selten

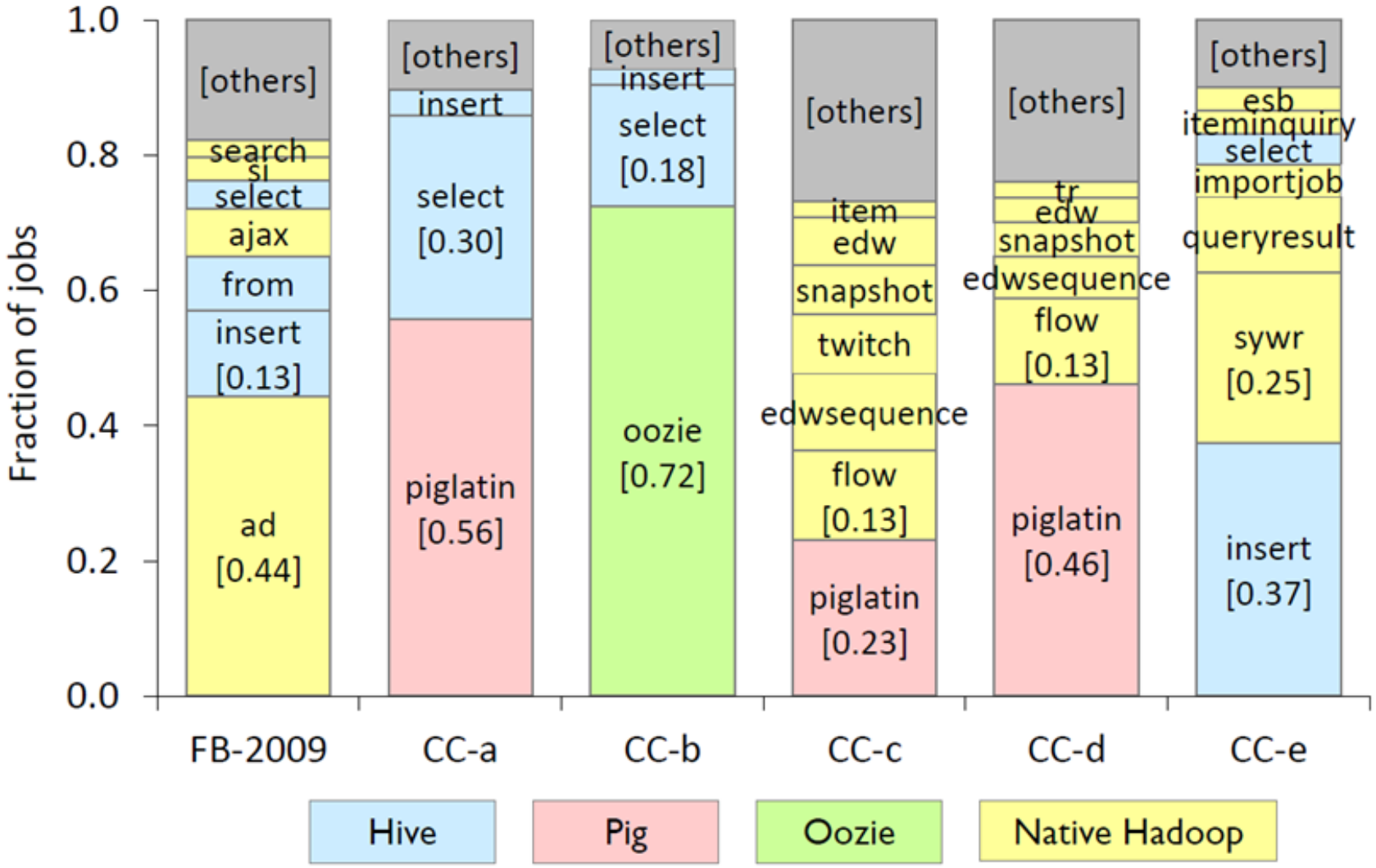


Quelle: Chen et. al: Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. VLDB 2012

Anforderungen für Ad-hoc-Datenanalyse

- Keine DWH-artige Vorverarbeitung, insbesondere für wenig genutzte Dateien
  - Zu großer Entwicklungs-Overhead (vgl. ETL)
  - Ineffizient für große Dateien
- High-Level-Sprache zur Definition von Analyseaufgaben
  - Analysten müssen nicht zwangsläufig MapReduce-Programme schreiben können
  - Effiziente Wiederverwendung von Skripten/Workflows für gleiche/ähnliche Tasks
- Hadoop-basierte Frameworks zur Datenanalyse
  - Hive: Automatische Übersetzung von SQL(-artigen) Statements in MapReduce-Programme
  - Pig: Ausführung prozeduraler „Datenflüsse“ (ähnlich SQL-Skript-Erweiterungen) auf Hadoop
  - Oozie: Management von Hadoop-Workflows (u.a. IF-THEN-ELSE, Schleifen)

Verbreitung von Hadoop-basierten Frameworks



Quelle: Chen et. al: Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. VLDB 2012