# MapReduce

Johann Volz

3. Juni 2010

**Zusammenfassung**

Bei der Verarbeitung von Datenmengen, die hunderte oder gar tausende Computer zur Fertigstellung in der gewünschten Zeit brauchen, muss man sich nicht nur mit der eigentlichen Verarbeitungsfunktion seines Programmes befassen, sondern man muss sich zusätzlich noch über die Verteilung dieser Rechenaufgabe auf die einzelnen Computer Gedanken machen.

Eine Abstraktion, die dies erleichtert, ist das MapReduce-Programmiermodell. Es ist inspiriert durch die Funktionen *map* und *reduce* des funktionalen Programmierens, auch wenn die Begriffe bei MapReduce unterschiedliche Bedeutung haben. Ein MapReduce-Programm besteht nur aus Map- und Reduce-Funktionen und deren Aneinanderreihung. Die Map-Funktion generiert ausschließlich aus dem zugeteilten Teil der Daten Schlüssel-Wert-Paare. Die Reduce-Funktion aggregiert danach alle Werte gleichen Schlüssels.

Da keiner der Map-Aufrufe von einem anderen abhängt und auch die Reduce-Funktion für jeden Schlüssel unabhängig berechnet werden kann, lassen sich nach dem MapReduce-Modell geschriebene Programme automatisch parallelisieren. Dies ist nur dadurch beschränkt, wie fein sich die Eingabedaten aufteilen lassen. Benutzt man ein MapReduce-Framework, so kann man sich auf die Kernfunktionalität seines Programmes konzentrieren.

## 1 Einleitung

MapReduce ist ein Programmiermodell, bei dem Programme nur in Form von *Map*- und *Reduce*-Funktionen ausgedrückt werden. Die Beschränkung darauf erlaubt ein automatisiertes Parallelisieren.

### 1.1 Die map- und reduce-Funktionen des funktionalen Programmierens

Inspiriert wurde MapReduce durch die Funktionen *map* und *reduce* beim funktionalen Programmieren [1], die auch in andere Programmiersprachen Einzug gefunden haben. Diese Funktionen sollen an dieser Stelle kurz erläutert werden, konkrete Beispiele beziehen sich auf die Implementierung dieser Funktionen in Python [2]. Die *map*-Funktion wendet eine als Parameter

angegebene Funktion $f$ auf alle Werte einer Liste an und gibt die resultierende Liste zurück. Die Funktion $f$ selbst erhält also einen einzelnen Wert aus der Liste und gibt als Ergebnis genau einen Wert als Resultat zurück. Die *reduce*-Funktion erhält eine Liste von Werten und reduziert diese anhand einer angegebenen Funktion $g$ auf einen einzigen Wert. Dazu übergibt sie an die Funktion $g$ als Argumente jeweils das letzte Ergebnis von $g$ und den nächsten Wert in der Liste. Für den ersten Aufruf von $g$ werden entweder die ersten beiden Elemente der Liste genommen oder ein benutzerdefinierter Startwert. Man kann die *reduce*-Funktion als Verschachtelung von $g$ auffassen, der Code

```
result = g(g(g(a, b), c), d)
```

ist also äquivalent zu:

```
result = reduce(g, [a, b, c, d]).
```

In Kombination lassen sich *map* für die Berechnung von Resultaten aus einzelnen Werten und *reduce* für Aggregation dieser Resultate verwenden. Will man etwa die Summe der Quadrate einer Liste berechnen, so wäre der imperative Ansatz vielleicht:

```
squareSum = 0
def add(x, y): return x + y
def square(x): return x**2
for number in [1, 2, 3, 4]: squareSum = add(squareSum, square(number))
```

Mit *map* und *reduce* ausgedrückt (und gleichen Funktionen *add* und *square*) würde man schreiben:

```
squareSum = reduce(add, map(square, [1, 2, 3, 4])).
```

## 1.2 MapReduce

Bei den im Folgenden beschriebenen abweichenden Definition beim MapReduce-Programmiermodell gegenüber dem funktionalen Programmieren ist insbesondere zu beachten, dass der Begriff *Map*-Funktion bei MapReduce eher der beim funktionalen Programmieren an die *map*-Funktion *übergebenen* Funktion entspricht. Mit der *Reduce*-Funktion bei MapReduce bezeichnet man ebenfalls die Funktion, die die Art der Aggregation festlegt. Auch wenn die Definitionen von *Map* und *Reduce* beim MapReduce-Programmiermodell anders sind als beim funktionalen Programmieren, bleibt die Grundidee erhalten: Mithilfe der *Map*-Funktion werden aus Eingabedaten Ergebnisse generiert und dann von der *Reduce*-Funktion aggregiert.

Die nachfolgenden Definitionen für *Map* und *Reduce* sind als unabhängig von den vorangegangen zu verstehen. Im Rest dieses Textes sind mit den Begriffen immer die Definitionen aus dem MapReduce-Programmiermodell gemeint.

Die *Map*-Funktion erhält ein Schlüssel-Wert-Paar als Eingabe. Bei der Aufteilung einer großen Menge an Textdokumenten könnte der Schlüssel etwa jeweils der Dateiname sein und der Wert der Inhalt der Datei. Die Map-Funktion generiert aus diesem Schlüssel-Wert-Paar an-

dere Schlüssel-Wert-Paare als Ausgabe. Die Typen der Paare der Eingabe und Ausgabe sind dabei unabhängig voneinander. Beim Beispiel der Textdateien könnte etwa ein vorkommendes Wort der Schlüssel sein und die Anzahl der Vorkommnisse der Wert. Die Zuordnung ist:

```
map: key, value -> key1, value1, key2, value2, ...
```

Die *Reduce*-Funktion erhält einen Schlüssel und eine Menge aller zu diesem Schlüssel gehörenden Werte. Beim gerade genannten Beispiel würde die Reduce-Funktion also ein bestimmtes Wort als Schlüssel erhalten und die Liste aller Zählungen der einzelnen Map-Abläufe als Werte. Die Reduce-Funktion generiert dann ein aggregiertes Ergebnis für diesen Schlüssel. Die Zuordnung ist:

```
reduce: key, value1, value2, ... -> result
```

Der grundsätzliche Ablauf eines MapReduce-Programms besteht aus der Aufteilung der Daten auf mehrere gleichartige Map-Abläufe, der Weiterleitung der Wertelisten für jeden von den Map-Funktionen generierten Schlüsseln an die Reduce-Abläufe und schließlich das Ausführen dieser Reduce-Funktionen, um ein Endergebnis zu erhalten. Auf jeden Fall vom Ersteller des MapReduce-Programms anzugeben sind die Map- und Reduce-Funktionen.

Da die Map-Funktion nur Informationen aus dem ihr zugeteilten Datenteil und die Reduce-Funktion nur die Werte zu dem ihr zugewiesenen Schlüssel benötigt, lassen sich sämtliche Map- und sämtliche Reduce-Aufrufe parallel durchführen. Um MapReduce-Programme auf einem Cluster laufen zu lassen, bedarf es einer Implementierung eines MapReduce-Frameworks. Das Framework muss nebenläufig mehrmals die Map-Funktion auf verschiedenen Computern aufrufen und dabei auch die Teile der Eingabedaten zuordnen. Es muss von diesen die Ergebnisse entgegennehmen und Listen von Werten mit gleichen Schlüsseln erzeugen. Für jeden Schlüssel muss nun wieder die Reduce-Funktion aufgerufen werden. Damit kann schon begonnen werden, auch wenn die Map-Aufrufe noch nicht abgeschlossen sind. Sind alle bisher berechneten Werte von der Reduce-Funktion bereits verarbeitet, muss diese warten, bis sie weitere Werte zu ihrem Schlüssel erhält oder alle Map-Aufrufe fertig sind.

Einige Beispiele, die die Darstellung von Programmen in Form von Map- und Reduce-Funktionen veranschaulichen sollen, sind in Abschnitt 2 beschrieben. Abschnitt 3 beschreibt die Implementierung und Benutzung von MapReduce bei Google. Mit Hadoop wird in Abschnitt 4 eine Google MapReduce ähnliche, aber freie Implementierung vorgestellt. Wie diese als Software-as-a-Service von Amazon angeboten wird, wird in Abschnitt 5 beschrieben.

## 2 Beispiele

An einigen Beispielen soll nun gezeigt werden, wie sich Auswertungsaufgaben in Form von Map- und Reduce-Funktionen darstellen lassen.

**Wörter in Dokumenten zählen** Es soll die Anzahl der Vorkommnisse jedes Worts in einer Menge von Textdokumenten gezählt werden [1]. Die *Map*-Funktion nimmt als Eingabe-Schlüssel etwa den Namen des aktuellen Dokuments; der Wert ist dessen Inhalt. Sie teilt diesen Inhalt in Wörter auf und gibt für jedes Wort W die Zwischendaten `Schlüssel: W, Wert: 1` aus. Die *Reduce*-Funktion addiert nun alle Werte zusammen und gibt für ihren Schlüssel K das Ergebnis `(K, Vorkommnisse von K)` aus. Dieses Beispiel zeigt auch, dass der ursprüngliche Schlüssel (Dokumentenname) irrelevant sein kann und nicht zwangsläufig von der Map-Funktion verwendet werden muss.

**Distributed Grep** Angelehnt an das Programm *Grep* sollen nun verteilte Textdaten nach Übereinstimmung mit einem regulären Ausdruck durchsucht werden [1]. Die *Map*-Funktion erhält wie im obigen Beispiele Dokumentennamen und -inhalt und überprüft letzteren, ob er mit dem regulären Ausdruck übereinstimmt. Ist dies der Fall, gibt sie die Position des Fundes als Schlüssel aus und als Wert etwa die tatsächlich gefundene Übereinstimmung. Die *Reduce*-Funktion ist die Identität. Es findet also nach dem Durchsuchen keine weitere Berechnung mehr statt.

**Durchschnittliches Einkommen in Städten** Dieses Beispiel ist komplexer und demonstriert die Möglichkeit, MapReduce-Jobs miteinander zu verketten. Aus zwei Datensätzen soll das Durchschnittseinkommen in Städten berechnet werden [3]. Ein Datensatz enthält pro Bürger die Sozialversicherungsnummer (SVN) und persönliche Informationen, unter anderem, in welcher Stadt er lebt. Der zweite Datensatz enthält die SVN und das Einkommen. Die Map-Funktion *Map 1a*, die den ersten Datensatz bearbeitet, gibt nun das Paar `Schlüssel: SVN, Wert: Stadt` aus. Die Map-Funktion *Map 1b*, die den anderen bearbeitet, gibt das Paar `Schlüssel: SVN, Wert: Einkommen` aus. Die Ergebnisse beider Map-Funktionen werden nun an die Reduce-Funktion *Reduce 1* weitergegeben, die die beiden Werte für jede SVN aggregiert. Nun nimmt *Map 2* diese Daten und gibt sie jeweils als Paar `Schlüssel: Stadt, Wert: Einkommen` aus. Schließlich fasst *Reduce 2* die Einkommen der gleichen Stadt zusammen und bildet den Durchschnitt.

# 3 MapReduce bei Google

## 3.1 Motivation

In ihrem Paper "MapReduce: Simplified Data Processing on Large Clusters" [1] beschreiben Jeffrey Dean und Sanjay Ghemawat ausführlich die von ihnen für Google entwickelte MapReduce-Implementierung und Infrastruktur und die Motivation, diese zu entwickeln.

Bei Google müssen sehr große Mengen an Daten verarbeitet werden. Die eigentliche Berechnungen, die mit den Daten angestellt werden müssen, sind dabei oft vergleichsweise einfach im Gegensatz zum programmiertechnischen Aufwand, diese Berechnungen auf viele Computer zu verteilen und die Ergebnisse wieder zusammenzutragen. Da dieser Aufteilungsvorgang vom Grundprinzip her aber immer gleich verläuft, ließ sich eine Abstraktion entwickeln, die diese

Aufgaben übernimmt und es dem Programmierer erlaubt, sich auf die eigentliche Funktionalität seines Programms zu konzentrieren. Hierfür wurde Google MapReduce geschaffen.

### 3.2 Implementierung

Googles MapReduce-Implementierung ist darauf ausgerichtet, im Datenzentrum zusammen mit dem verteilten Google File System (GFS) [4] eingesetzt zu werden. Bei GFS werden alle Daten redundant auf mehreren Computern gespeichert und nicht etwa in einzelnen großen Rechnern mit RAID. Dies macht sich Google MapReduce bei der Verteilung der Aufgaben auf die Computer zunutze, indem es den Computern diejenigen Daten zuweist, die dort schon gespeichert sind. Dadurch lässt sich der Datentransfer zwischen den Computern im Rechenzentrum reduzieren. Im Idealfall müssen sogar ausschließlich Metadaten übertragen werden.

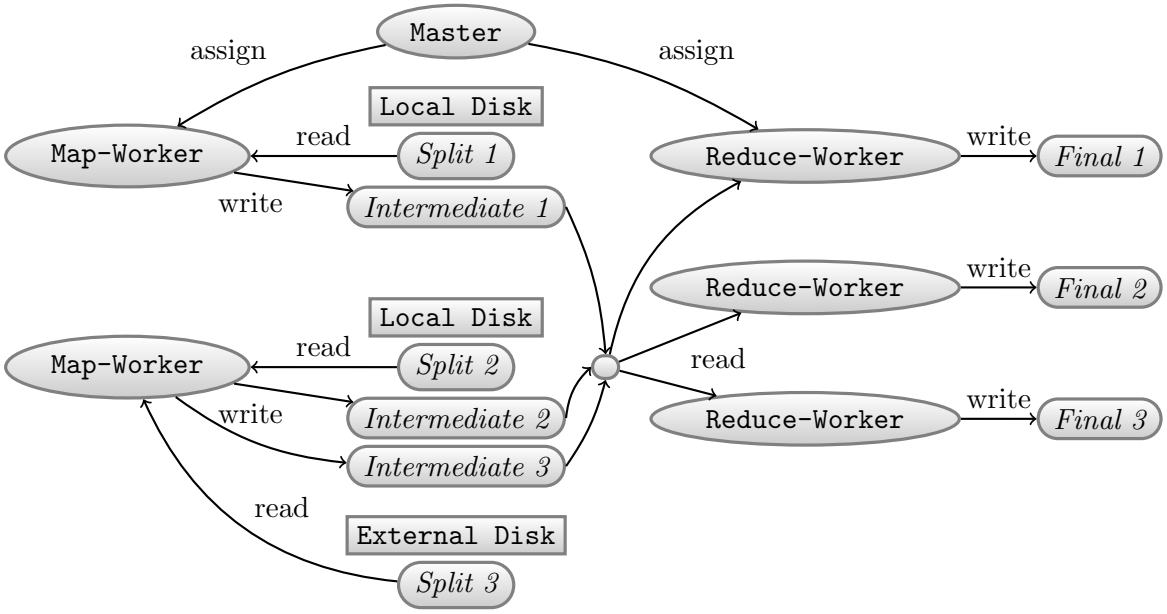


Abbildung 1: Die Architektur von Google MapReduce

Abbildung 1 zeigt die Architektur und den Ablauf eines MapReduce-Jobs. Der Master bestimmt zunächst die als Map-Worker einzusetzenden Computer und teilt die Daten zwischen ihnen auf. Typische Größen für einen Teil der Eingabedaten sind 16 bis 64 Megabytes. Er weiß dabei, welche Eingabedaten auf welchem Computer gespeichert sind und wo sich die Computer innerhalb der Netzwerktopologie befinden (z.B. welche Computer sich im selben Rack befinden). Wann immer möglich wird er Map-Jobs den Computern zuweisen, die auf ihren lokalen Festplatten die zugehörigen Daten haben oder zumindest ein Rack mit dem Computer teilen, der diese Daten hat. Die Map-Worker führen ihre Funktion aus und schreiben die Zwischenwerte auf ihre lokale Festplatte und teilen dem Master die Position dieser Daten mit. Der Master startet gleichzeitig auch die Reduce-Worker und teilt ihnen die Positionen ihrer Eingabedaten auf den Map-Workern mit. Diese Daten werden über das Netzwerk gelesen. Die Reduce-Worker führen nun ihre Funktion aus und schreiben die endgültigen Daten ins GFS.

Als optionale Komponente lässt sich bei Google MapReduce auch eine *Combine*-Funktion defi-

nieren, die die auf einem Map-Worker entstandenen Werte gleichen Schlüssels zusammenfasst. Die Combine-Funktion lässt sich als lokale Reduce-Funktion, die nur einen Teil der Werte zu einem Schlüssel erhält, auffassen. Beim Wörterzählbeispiel etwa lassen sich n Paare der Form `Schlüssel: BestimmtesWort, Wert: 1` zu einem einzigen Paar `Schlüssel: BestimmtesWort, Wert: n` zusammenfassen. Dies spart die Bandbreite, von einem Map-Worker Paare mit dem gleichen Schlüssel einzeln zu übertragen. Die Zuordnung ist hier:

`combine: key1, value1, key2, value2, ... -> key, value`

### 3.3 Fehlertoleranz

Es kann sein, dass das vom Programmierer vorgegebene Programm für bestimmte Teile der Eingabedaten fehlerhaft ist, beziehungsweise dass die Eingabedaten selbst Fehler aufweisen. Außerdem ist aufgrund der Anzahl der an einem MapReduce-Job beteiligten Maschinen ein Ausfall einer oder mehrerer dieser Computer wahrscheinlich genug, um eine automatisierte Reaktion darauf wünschenswert zu machen. Beide Fälle werden bei Google MapReduce vom Framework erkannt und Maßnahmen ergriffen.

Um festzustellen, ob ein Worker ausgefallen ist, pingt der Master regelmäßig alle von ihm zugeteilten Worker an. Wird eine Zeitgrenze für die Antwort überschritten, so wird von einem fehlerhaften Worker ausgegangen. Der Master teilt nun die einst an diesen Worker vergebenen Aufgaben neu zu. Bei Map-Workern gilt insbesondere, dass die Aufgabe auch dann neu zugeteilt werden muss, wenn sie bereits abgeschlossen war, da die Zwischendaten auf der lokalen Festplatte dieses Workers gespeichert wurden. Ausfälle des Masters sind so selten, dass dieser Ausfall nicht automatisch kompensiert wird.

Auch Fehler im MapReduce-Programm oder in den Eingabedaten werden vom Master erkannt. Stürzt ein Worker ab, so sendet er vorher noch eine Statusinformation mit der Meldung über den Absturz und den ihn verursachenden Teil der Daten. Werden beim gleichen Teil der Daten mehrmals Abstürze gemeldet, so wird dieser nicht erneut zugeteilt, sondern übersprungen. Würde man diese Fehlererkennung weglassen, würden fehlerhafte Daten zu endlosen Neu-Zuteilungen von Aufgaben führen.

Um kurz vorm Ende eines MapReduce-Jobs die Wartezeit auf die Fertigstellung zu verkürzen, werden die verbleibenden Aufgaben auf weiteren Maschinen ausgeführt. Dadurch können Verzögerungen, die durch langsame oder durch andere Jobs überlastete Computer entstehen, ausgeglichen werden und der Job zu einem zügigen Ende gebracht werden.

### 3.4 Einsatzgebiete von MapReduce bei Google

MapReduce übernimmt bei Google den Großteil der Berechnungen, es ist “Googlers’ hammer for 80% of our data crunching” [3]. Beispielsweise wird der für die Websuche erforderliche Index durch eine Folge von fünf bis zehn MapReduce-Jobs aus den zuvor im GFS gespeicherten Webseitendaten generiert [1]. Es wird zudem bei Berechnungen für Machine-Learning, Clustering-Probleme für Google News, Auswertung von Suchanfrage-Logs für Google Zeitgeist [1] und das

Zusammenfügen von Satellitendaten verwendet [3].

# 4 Hadoop MapReduce

## 4.1 Beschreibung

Hadoop MapReduce ist eine freie Implementierung eines MapReduce-Frameworks, deren Funktionalität stark der der von Google beschriebenen Architektur [1] ähnelt, insbesondere die enge Verknüpfung mit einem verteilten Dateisystem. Hadoop bietet hierfür das Hadoop Distributed File System (HDFS) [5].

Hadoop MapReduce zieht ebenso wie Google MapReduce das Verschieben der Berechnung zu den Daten dem umgekehrten Fall vor und bezieht die Speicherorts-Informationen aus HDFS mit in seine Aufgabenverteilung ein. Es bietet zudem auch Fehlertoleranz gegenüber Computerausfällen und fehlerverursachenden Datenteilen [6, 7].

Hadoop-MapReduce-Programme können in Java durch die Implementierung der Mapper- und Reducer-Interfaces geschrieben werden. Es ist aber durch das sogenannte *Streaming* auch möglich, ein beliebiges Programm, welches von der Standardeingabe lesen und in die Standardausgabe schreiben kann, als Mapper und Reducer einzusetzen [7]. Auch die Angabe von *Combine*-Funktionen wie bei Google MapReduce ist möglich.

Hadoop ist aufgrund seines Framework-Overheads für Cluster unter einem Dutzend Computer ineffizient. Die Vorteile der vereinfachten Verteilung von Jobs gegenüber manuell implementierter Kommunikation wie etwa mit MPI zeigt sich auch erst bei einer größeren Menge an Computern [6].

Neben dem verteilten Betrieb bietet Hadoop auch die Möglichkeit an, MapReduce-Jobs lokal im Zusammenspiel mit dem normalen Dateisystem auszuführen. Dies erfordert nur einen geringfügigen Konfigurationsaufwand und erlaubt es einem, Hadoop-MapReduce-Programme vor dem Einsatz auf einem Cluster mit einer kleinen Datenmenge zu testen und zu debuggen [7].

## 4.2 Einsatz von Hadoop

Der größte Einsatz von Hadoop ist bei Yahoo!, wobei gespeicherte Webseiten in einem Cluster mit 10.000 Cores für die spätere Verwendung in der Websuche analysiert werden. Hierbei werden etwa eine Billion Links analysiert und 300 Terabytes an komprimierten Ausgabedaten generiert [8]. Desweiteren wird Hadoop noch bei AOL, Hulu, ImageShack, Facebook, Last.fm und Twitter eingesetzt [9].

# 5 Amazon Elastic MapReduce

## 5.1 Elastic Compute Cloud und Simple Storage Service

Zum Verständnis von Elastic MapReduce müssen zunächst zwei andere von Amazon Web Services angebotene Dienste erläutert werden. Zum einen ist dies die Elastic Compute Cloud (EC2). Bei EC2 kann der Benutzer auf Bedarf sofort Rechenkapazität in Form von virtuellen Maschinen (Instanzen) unterschiedlicher Leistung anfordern. Abgerechnet wird nach benutzten Maschinenstunden. Ebenfalls von Amazon angeboten wird der Simple Storage Service (S3). Bei S3 kann ein Benutzer beliebige Dateien in sogenannten Buckets – nicht-hierarchischen Ordnern – speichern, diese wieder selbst abrufen oder im Web zur Verfügung stellen. S3 übernimmt dabei das redundante Speichern und zuverlässige Anbieten der Daten. Abgerechnet wird nach benutztem Speicherplatz und Traffic.

## 5.2 Elastic MapReduce

Amazon Elastic MapReduce (EMR) baut auf den beiden beschriebenen Diensten EC2 und S3 auf. EMR bietet Hadoop als Dienst auf EC2-Instanzen an, die ihre Daten von S3 lesen. Der Benutzer muss dafür zunächst die Eingabedaten und das MapReduce-Programm nach S3 hochladen. Dann kann der Benutzer über ein Webinterface seinen MapReduce-Job konfigurieren. Hier gibt er den Mapper und Reducer und die S3-URL der Eingabedaten an. Da EMR Hadoop verwendet, sind hier alle Programme möglich, die Hadoop auch akzeptiert. Es können also sowohl Jar-Dateien mit Mapper- und Reducer-Klassen verwendet werden, als auch beliebige Programme als Mapper und Reducer angegeben werden. Zudem wird ein weiterer S3-Bucket festgelegt, in dem die Ergebnisdaten gespeichert werden sollen. Im nächsten Schritt werden die Anzahl und Art der EC2-Instanzen, die den Job bearbeiten sollen, festgelegt. Während des Verlaufs des MapReduce-Jobs wird der Status im Webinterface angezeigt. Das Starten und Beobachten von Jobs ist auch automatisiert über eine API möglich [10].

Abbildung 2 zeigt die Architektur von Amazon Elastic MapReduce. Eine der EC2-Instanzen fungiert als Master und teilt den restlichen Instanzen Aufgaben zu. Die Map-Worker laden sich die benötigten Dateien von S3 herunter und speichern ihre Zwischendaten auf ihrer lokalen Festplatte. Die Reduce-Worker lesen diese von dort wieder aus und laden am Ende das Ergebnis zu S3 hoch.

Der Nachteil von Elastic MapReduce gegenüber den Implementierungen von Hadoop mit HDFS oder Google ist die fehlende Datenlokalität, da die Daten in jedem Fall von S3 über das Netzwerk auf die EC2-Instanzen übertragen werden müssen, statt dass von der lokalen Festplatte gelesen werden kann.

Der Vorteil ist, dass keinerlei Einrichtungsaufwand für einen eigenen Hadoop-Cluster entsteht. Zudem muss keine für den MapReduce-Job ausreichend große Hardwarekapazität dauerhaft bereitgehalten werden.

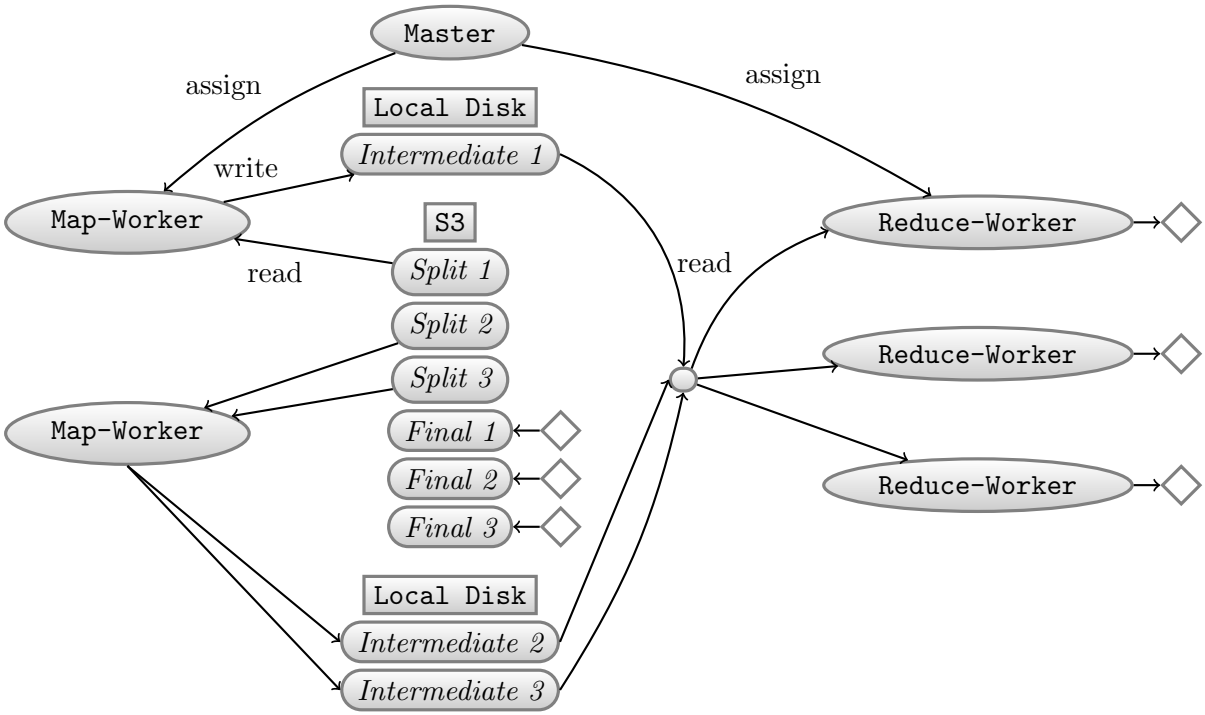


Abbildung 2: Die Architektur von Amazon Elastic MapReduce

### 5.3 Einsatz von Elastic MapReduce

Die amerikanische Zeitung *The New York Times* nutzte Amazon Elastic MapReduce um 4 TB an in Bildform vorliegender alter Zeitungsartikel in PDFs umzuwandeln. Dazu wurden 100 EC2-Instanzen verwendet, die die Aufgabe in unter 24 Stunden erledigten [11]. Bei der Dating-Seite eHarmony werden Match-Making-Algorithmen mithilfe von EMR ausgeführt [12]. Die Online-Werbefirma *Razorfish* benutzt EMR, um Daten über Nutzerverhalten zu analysieren [13].

## 6 Fazit

MapReduce ist ein geeignetes Programmiermodell für eine Vielzahl von Aufgaben. Wie jede Abstraktion erlaubt es dem Programmierer, sich auf die spezielle Funktionalität seines Programmes zu konzentrieren, statt sich mit darunterliegenden und für viele Programme gleichen Implementierungsdetails auseinandersetzen zu müssen. MapReduce-Programme lassen sich ähnlich leicht schreiben wie für den Betrieb auf einem einzigen Computer bestimmte.

Die vorhandenen Implementierungen bringen zudem bedeutende Optimierungen und Robustheit mit sich. So behebt der Umgang mit ausgefallener Hardware eine Fehlerklasse, die man bei traditionellen Programmen für einzelne Computer für gewöhnlich gar nicht betrachtet. Zudem bieten die Implementierungen von Google mit GFS und Hadoop mit HDFS gute Datenlokalität und bringen damit für datenintensive Berechnungen eine Performance-Steigerung und Verringerung der benötigten Bandbreite. Dies erfordert allerdings auch eine spezielle Rechenzentrums-

Architektur, etwa, dass die Datenhaltungs- und Berechnungsleistung von den gleichen Rechnern gestellt wird. Ist man aber bereit, auf den Vorteil der Datenlokalität zu verzichten, so ist MapReduce in Form des Amazon-Dienstes auch dann interessant, wenn man nicht bei Google arbeitet oder ein Rechenzentrum betreibt. Insbesondere ist hier die Möglichkeit, viele Instanzen rein nach Nutzungszeit abgerechnet zu verwenden, interessant. So lohnt sich auch dann das Schreiben eines parallelisierbaren MapReduce-Programms, wenn man bei der Alternative, dem Verwenden eigener, dauerhaft bereitgehaltener Hardware vielleicht nur einen einzigen Computer nehmen würde.

So wie Virtual Memory die Speicheraufteilung, Betriebssysteme die Hardware und Hochsprachen die Maschinensprache abstrahieren, so ist MapReduce eine bedeutende Abstraktion für verteilten Ablauf von Programmen.

## Literatur


[1] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI Conference* (2004)

[2] *Python v2.6.5 documentation - Built-in Functions.* `http://docs.python.org/library/functions.html`

[3] Zhao, Jerry ; Pjesivac-Grbovic, Jelena: *MapReduce - The Programming Model and Practice.* 2009

[4] Ghemawat, Sanjay ; Gobioff, Howard ; Leung, Shun-Tak: The Google File System. In: *SOSP* (2003)

[5] Borthakur, Dhruba: *The Hadoop Distributed File System: Architecture and Design.* 2007

[6] Yahoo!: *Hadoop Tutorial - Yahoo! Developer Network.* `http://developer.yahoo.com/hadoop/tutorial/`

[7] Apache Software Foundation: *Hadoop Map/Reduce Tutorial.* `http://hadoop.apache.org/common/docs/current/mapred_tutorial.html`. Version: 2010

[8] Baldeschwieler, Eric: *Yahoo! Launches World's Largest Hadoop Production Application.* `http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html`. Version: Februar 2008

[9] *Hadoop Wiki: PoweredBy.* `http://wiki.apache.org/hadoop/PoweredBy`

[10] *Amazon Web Services - Documentation.* `http://aws.amazon.com/documentation/`

[11] Gottfrid, Derek: Self-service, Prorated Super Computing Fun! In: *New York Times* (2007), November

[12] Lawton, George: *eHarmony puts Amazon MapReduce to the test.* `http://searchcloudcomputing.techtarget.com/news/article/0,289142,sid201_gci1358755,00.html`. Version: Juni 2009

[13] *AWS Case Study: Razorfish.* `http://aws.amazon.com/solutions/case-studies/razorfish/`