VERY LARGE
BUSINESS APPLICATIONS
Carl von Ossietzky Universität Oldenburg

Hadoop / Hive

Referat
im Rahmen des Modul Business Intelligence 2

Themensteller:	Prof. Dr.-Ing. Jorge Marx Gómez
Betreuer:	M.Sc. Stefan Wunderlich

Vorgelegt von:	Nils Lutz Semesteranschrift PLZ Wohnort: Telefonnummer: mustermann@uni-oldenburg.de

Abgabetermin:	25. August 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Text

2. Hadoop / Hive

Dieses Kapitel soll zu erst eine Übersicht über das Hadoop Framework im allgemeinen geben. Dabei wird auf die wichtigsten Komponenten aus dem Framework eingegangen und im Anschluss folgt eine Erläuterung des von Hadoop verwendeten Dateisystems HDFS. Außerdem wird der MapReduce Algorithmus im Detail betrachtet.

2.1. Hadoop Framework

Das Apache Hadoop Framework ist ein, in Java geschriebenes, freies Framework für verteilt arbeitende Software. Es ist eine konkrete Implementierung des MapReduce-Algorithmus. Mit Hadoop sind intensive Berechnungsaufgaben über große Datenmengen im Petabyte-Bereich auf Clustern möglich. Es können herkömmliche Heimcomputer verwendet und als Cluster eingerichtet werden. Um die Funktionalität bereitzustellen nutzt Hadoop einerseits das eigens dafür entwickelte Dateisystem Hadoop Distributed File System und andererseits den MapReduce-Algorithmus.(vgl. [Wik15a])
Betrieben werden kann Hadoop auf den geläufigen Unix/Linux Distributionen. Unter Windows läuft Hadoop hingegen nur mit Cygwin. Anwendungen die für oder auf Hadoop basieren können in den verschiedensten Programmiersprachen entwickelt werden. Darunter sind beispielsweise Java, Python und C++ um nur ein kleinen Teil zu nennen. Hadoop selbst benötigt mindestens Java 1.6. Neben dem eigenen Dateisystem HDFS werden noch weitere andere Storagemöglichkeiten wie Amazon S3, CloudStore oder Kosmos unterstüzt.

HBase

HBase bildet eine skalierbare Datenbank für sehr große Datenmengen innerhalb des Hadoop Framework. HBase wurde als eine Implementierung von Google BigTable entwickelt. Der Gegensatz von Google BigTable zu relationalen Datenbanken besteht darin, dass die Anzahl der Spalten nicht für eine gesamte Tabelle bestimmt wird. Bei Google BigTable kann die Anzahl der Spalten je Zeile variieren. Außerdem wird jede Zelle innerhalb einer Tabelle mit einem eigenen Zeitstempel versehen und kann dadurch sehr einfach versioniert werden.

Hive

Hive ist eine auf SQL basierende Abfragesprache. Ursprünglich wurde Hive von Facebook Inc. entwickelt steht aber seit 2008 als Open-Source-Version verfügbar. Der Nutzer kann mit Hive Tabellen und Spalten definieren und einen MapReduce Prozess starten, welcher

diese Tabellen und Spalten dann nutzt. Mit HiveQL, so der Name der Abfragesprache, lassen sich Zusammenfassungen, Berichte und Analysen erzeugen.(vgl. [Bio10])

Pig

Pig ordnet sich im Hadoop Framework als High-Level Datenflusssprache ein. Mit Pig lassen sich einfach MapReduce Programme entwickel die auf einem Hadoop System ausgeführt werden können. Pig charakterisiert sich durch folgende Eigenschaften:

- „Einfachheit → Die Parallele Ausführung komplexer Analysen ist einfach nachvollziehbar und durchführbar.
- Optimierung → Pig optimiert selbständig die Ausführung komplexer Operationen nach der Carsten-Methode.
- Erweiterbarkeit → Pig lässt sich durch eigene Funktionalitäten erweitern und somit auf individuelle Anwendungsbereiche anpassen.“(frei übersetzt von [Fou15])

2.2. Hadoop Distributed File System

Text

2.3. MapReduce Algorithmus

MapReduce ist ein von Google Inc. entwickeltes und eingeführtes Programmiermodell. Es wurde speziell zur nebenläufigen Berechnung über große Datenmengen auf Computerclustern entworfen. MapReduce nutzt dabei mehrere Phasen - Map, Shuffle, Reduce - um die Daten zu verarbeiten. Dabei können zwei der Phasen durch den Anwender parametrisiert werden. So lassen sich die Berechnungen parallelisieren und über mehrere Computer verteilen. In einigen Fällen ist eine Parallelisierung alleine durch die großen Datenmengen unumgänglich, da ein einzelner Prozess mit der Verarbeitung überfordert wäre.
MapReduce nutzt dafür eine verteilte Speicherung der Daten in Blöcken. Dies sorgt für Datenqualität bei fehlerhaften Schreib- oder Lesevorgängen. Des Weiteren lässt sich MapReduce mit handelsüblichen Computer verwenden. Es ist keine spezielle Server Hardware nötig.(vgl. [Wik15b])

Definition der *MapReduce* Funktion

Die *MapReduce* Funktion baut sich aus zwei separaten Abläufen, *Map* und *Reduce*, auf. Der Input wird aus einer Menge unstrukturierten oder semi-strukturierten Daten gebildet. Die Basis der *Map* und *Reduce* Abläufe ist die Bildung von Schlüssel-Wert-Paaren. Es wird aus einer Liste von Schlüssel-Wert-Paraen eine neue Liste von Schlüssel-Wert-Paaren. Die sich daraus ergebenden Formeln lauten wie folgt:

$$MapReduce : (K \times V)^* \to (L \times W)^*$$
$$[(k_1, v_1) \ldots, (k_n, v_n)] \to [(l_1, w_1) \ldots (l_n, w_n)]$$

- Die Mengen K und L enthalten Schlüssel, die Mengen V und W enthalten Werte.
- Alle Schlüssel $k \in K$ sind vom gleichen Typ, z. B. Strings.
- Alle Schlüssel $l \in L$ sind vom gleichen Typ, z. B. ganze Zahlen.
- Alle Werte $v \in V$ sind vom gleichen Typ, z. B. Atome.
- Alle Werte $w \in W$ sind vom gleichen Typ, z. B. Gleitkommazahlen.
- Wenn A und B Mengen sind, so ist mit $A \times B$ die Menge aller Paare (a, b) gemeint, wobei $a \in A$ und $b \in B$ (kartesisches Produkt).
- Wenn M eine Menge ist, so ist mit M^* die Menge aller endlichen Listen mit Elementen aus M gemeint (angelehnt an den Kleene-Stern) - die Liste kann auch leer sein.

Die eigentliche Parametrisierung durch den Nutzer wird durch die beiden Funktionen *Map* und *Reduce* ermöglicht.

$$Map : K \times V \to (L \times W)^*$$
$$(k, v) \to [(l_1, x_1) \dots (l_r k, x_{rk})]$$

$$Reduce : L \times W^* \to X^*$$
$$(l, [y_1, \dots, y_{sl}]) \to [w_1, \dots, w_{ml}]$$

Im Anhang ist eine Beispiel Implementation der MapReduce Funktion zu finden.

Map Phase

Die Daten sind am Anfang partitioniert und repliziert im Hadoop Cluster. Jeder Rechner liest die lokalen Daten zeilenweise als Schlüssel-Wert-Paare und übergibt sie dem *Map*-Task (Implementation der *Map* Funktion auf einer Maschine). Der Schlüssel dieser Paare ist meistens der Byte-Offset einer Zeile und der Wert ist die Zeile selbst.Die Ausführung der *Map* Funktion auf jedes gelesene Paar erzeugt ein anderes temporäres Schlüssel-Wert-Paar. Außerdem sortiert der *Map*-Task seine generierten Paare nach Schlüssel und speichert sie temprär lokal. Es folgt eine Umverteilung der kompletten Ausgaben der Map Phase, indem die temporären Paare mit demselben Key aus allen *Map*-Tasks zum selben *Reduce*-Task gesendet werden.

Shuffle Phase

Bevor die Reduce Phase starten kann, müssen die Ergebnisse der Map Phase gruppiert werden anhand ihrer Schlüssel. Dazu ist ein koordinierter Datenaustausch nötig. Die Performance der gesamten *MapReduce* Funktion hängt maßgeblich von der Geschwindigkeit des Datenaustausch während der Shuffle Phase ab.

Reduce Phase

Wurden alle Ergebnisse der Map Phase gruppiert werden die Paare des Zwischenergebnisses an den *Reduce*-Task übergeben. Die *Reduce* Funktion erzeugt schließlich ein aggregiertes Paar. Wie auch in der Map Phase können in der Reduce Phase die einzelnen *Reduce*-Tasks über mehrere Rechner innerhalb des Clusters verteilt werden.

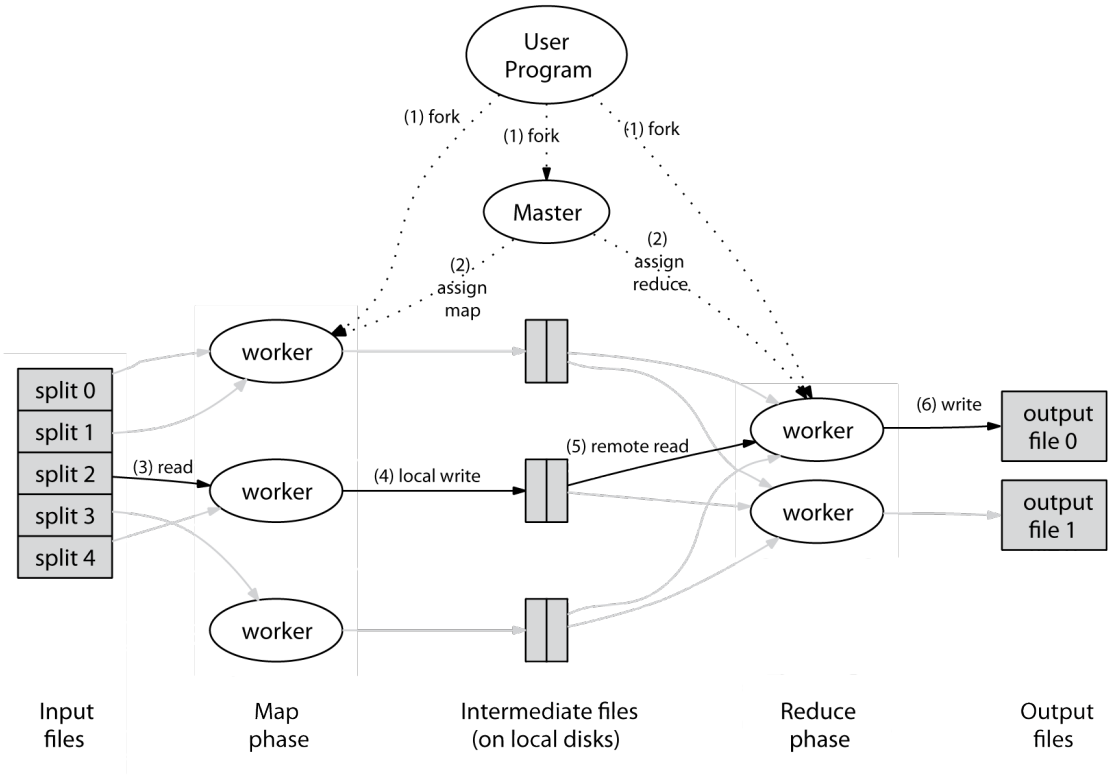

Abbildung 1: Ausführungsreihenfolge von MapReduce[DG04]

3. Fazit

// Resümee
// Ausblick

A. Anhang

```
1  import com.mongodb.*;
2
3  public class MongoClient {
4    public static void main(String[] args) {
5      Mongo mongo;
6      try {
7        DBCollection books = new Mongo("localhost", 27017).getDB("library").
             getCollection("books");
8
9        BasicDBObject book = new BasicDBObject();
10       book.put("name", "Understanding JAVA");book.put("pages", 100);
11       books.insert(book);
12
13       book = new BasicDBObject();
14       book.put("name", "Understanding JSON");book.put("pages", 200);
15       books.insert(book);
16
17       String map = "function() { "+
18           "var category; " +
19           "if (this.pages >= 250) category = 'Big Books'; " +
20           "else category = 'Small Books'; "+
21           "emit(category, {name: this.name});}";
22
23       String reduce = "function(key, values) { " +
24           "var sum = 0; " +
25           "values.forEach(function(doc) {sum += 1;}); " +
26           "return {books: sum};} ";
27
28       MapReduceCommand cmd = new MapReduceCommand(books, map, reduce,
29       null, MapReduceCommand.OutputType.INLINE, null);
30
31       MapReduceOutput out = books.mapReduce(cmd);
32
33       for (DBObject o : out.results()) {
34           System.out.println(o.toString());
35       }
36     } catch (Exception e) {e.printStackTrace();}
37   }
38 }
```

Listing 1: MapReduce in Java

Literatur

[Bio10] BMC Bioinformatics. *An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics*. 2010.

[DG04] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004.

[Fou15] The Apache Software Foundation. *Apache Pig*. 2015.

[Wik15a] Wikipedia. *Apache Hadoop*. 2015.

[Wik15b] Wikipedia. *MapReduce*. 2015.

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Diplomarbeit / Individuelle Projekt ...(Titel der Arbeit)... selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 22. August 2015 Nils Lutz; Janick Asmuß