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	Jannick Asmuss
Semesteranschrift:	Erlenweg 5	Erlenweg 5
PLZ Wohnort:	26129 Oldenburg	26129 Oldenburg
Telefonnummer:	+49 173 25 28 407	+49 174 19 32 947
Email:	Nils.Lutz@uni-oldenburg.de	Jannick.Asmuss@uni-oldenburg.de
Abgabetermin:	23. August 2015	

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Apache Hadoop ist ein Framework für skalierbare, verteilt arbeitende Software. Es ermöglicht unter Anderem effiziente Big-Data-Analysen. Das Framework ist allerdings eine relativ neue Technologie (vgl. [anG15]). Im Rahmen des Moduls „Business Intelligence II“ wird dieses Thema bearbeitet. Das Ziel dieser Ausarbeitung ist das Erarbeiten des aktuellen Standes der Technik. Außerdem soll bestimmt werden, aus welchen Komponenten das Hadoop Framework besteht. Die Ausarbeitung ist so strukturiert, das als Erstes, das Hadoop Framework bearbeitet wird. Dabei wird analysiert aus welchen Komponenten Hadoop besteht. Hadoop verwendet ein bestimmtes Dateiystem zum Arbeiten. Das Hadoop Distributed File System wird im zweiten Unterkapitel beschrieben. Im dritten Unterkapitel wird das MapReduce-Verfahren behandelt. Dabei soll geklärt werden, wie das Verfahren funktioniert. Im abschließenden Fazit werden die Ergebnisse der Ausarbeitung betrachtet.

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 einen kleinen Teil zu nennen. Hadoop selbst benötigt mindestens Java 1.6. Neben dem eigenen Dateisystem HDFS werden noch weitere andere Storage-Möglichkeiten wie Amazon S3, CloudStore oder Kosmos unterstützt.

HBase

HBase bildet eine skalierbare Datenbank für sehr große Datenmengen innerhalb des Hadoop Framework. HBase wurde als eine Implementierung von $GoogleBigTable$ entwickelt. Der Gegensatz von $GoogleBigTable$ zu relationalen Datenbanken besteht darin, dass die Anzahl der Spalten nicht für eine gesamte Tabelle bestimmt wird. Bei $GoogleBigTable$ 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 und ist 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 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 entwickeln 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. $HadoopDistributedFileSystem$

Das $HadoopDistributedFileSystem$ ist ein Bestandteil von Hadoop. In diesem Unterkapitel werden die Grundlagen dazu erläutert. Das File System wird definiert und kategorisiert. Außerdem werden die Ziele vom $HDFS$ genannt. Abschließend wird die Architektur erläutert.

Definition

Das *HadoopDistributedFileSystem* wird definiert als ein „hochverfügbares Dateisystem zur Speicherung sehr großer Datenmengen auf den Dateisystemen mehrerer Rechner (Knoten). Dateien werden in Datenblöcke mit fester Länge zerlegt und redundant auf die teilnehmenden Knoten verteilt. Dabei gibt es *Master*- und *Slave*-Knoten. Ein *Master*-Knoten, der so genannte *NameNode*, bearbeitet eingehende Datenanfragen, organisiert die Ablage von Dateien in den *Slave*-Knoten und speichert anfallende Metadaten. *HDFS* unterstützt dabei Dateisysteme mit mehreren 100 Mio. Dateien.“([Wik15b])

Distributed File Systems

Das *HadoopDistributedFileSystem* ist ein verteiltes Dateisystem. Ein verteiltes Dateisystem ist ein spezielles Dateisystem, mit dem der Zugriff auf Dateien über ein Rechnernetz erfolgt und das Zugriff und Dateienspeicherung auf mehreren als Server eingesetzten Rechnern erlaubt. Das Gegenstück zu solch einem Netzwerk-Dateisystem ist ein klassisches lokales Dateisystem, welches unmittelbar an den Computer angeschlossene Massenspeicher verwaltet. (vgl. [Wik15d])

Aufgaben von *HDFS*

Normale *DistributedFileSystems* können in bestimmten Situationen unpraktisch sein. Die Dateien lagern auf einer Maschine. Es können nur so viele Dateien auf einer Maschine gespeichert werden, wie die Maschine Speicher frei hat. Ist die Maschine nicht erreichbar, weil es z.B. Netzwerkprobleme gibt, können die Clients keine Dateien aufrufen. Außerdem kann der Zugriff von vielen Clients den Server überfordern. Das *HDFS* wurde so konzipiert, dass die Probleme anderer *DistributedFileSystems*, beim Einsatz von *HDFS* irrelevant sind. *HDFS* wurde für diesen Einsatzzweck kreiert. Es ist optimiert für die Speicherung von sehr großen Datenmengen von Informationen. Auch Datenmengen im Bereich von Terabytes oder Petabytes können verarbeitet werden. Die Daten werden über mehrere Maschinen verteilt. Es werden größere Dateiformate als bei normalen verteilten Dateisystemen unterstützt. Die Daten sind auch bei Ausfällen zuverlässig geschützt. Fällt ein Server aus, übernimmt der nächste Sever die Funktion. Das File System skaliert mit der Datenmenge. Werden höhere Datenmengen von vielen Clients angefordert, können weitere Rechner zum Rechnernetz hinzugefügt werden. (vgl. [Inc15])
Hardware-Defekte passieren regelmäßig. Ein *HDFS* kann aus mehr als 1000 Maschinen bestehen, wovon jede ein Teil des Dateisystems speichert. Da bei der Anzahl von Maschinen anzunehmen ist, dass nicht jede Maschine problemlos funktionieren wird, verfügt *HDFS* dafür über eine automatische Wiederherstellungsfunktion. Das *HadoopDistributedFileSystem*

ist ausgerichtet für Batchjobs statt für interaktiven Benutzerzugriff. Der Fokus liegt auf einem hohen Datendurchsatz statt auf einer niedrigen Latenz. (vgl. [Bor15])

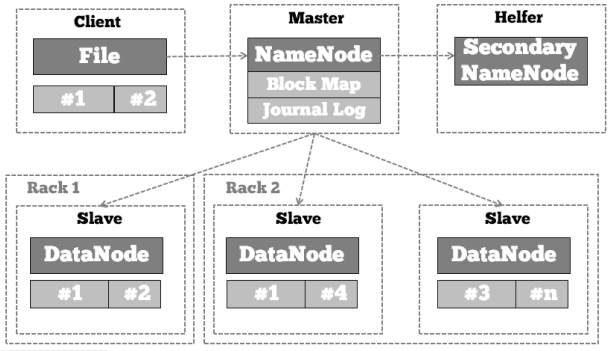

Abbildung 1: Architektur[Sel15a]

Architektur

Im Folgenden wird die Architektur von *HDFS* beschrieben. Die Architektur ist ausgelegt für Anwendungsfälle, wo Daten einmal ins File System übergeben werden und von dort vielfach ausgelesen werden. Damit die Daten verteilt gespeichert werden können, werden die Daten in Blöcke aufgespalten und auf den Knoten des Hadoop-Clusters verteilt. Für jeden dieser Blöcke werden Kopien auf verschiedenen Servern im Cluster angelegt, um Fehler und Ausfallsicherheit zu garantieren. Das zugehörige Management wird von *HDFS* übernommen. Der Zugriff auf das *HDFS* ähnelt Unix. Sowohl Konsolen-Befehle, als auch die Rechteverwaltung der Dateien funktionieren wie bei Unix.
Die Architektur ist eine *Master-Slave*-Architektur (Siehe Abbildung:1). Die Architektur umfasst dabei verschiedene Komponente, wie *Master*, *Slave*, *Client*, *NameNode* und *DataNode*. Der *Client* fordert eine Datei an vom Server. Der Server ist der *Master*. Der *Master* beinhaltet ein *NameNode*. Das *NameNode* ist die zentrale *Master*-Komponente und verwaltet die Metadaten für alle gespeicherten auf allen *DataNodes*. Im *BlockMap* speichert der Master alle aktuellen Speicherorte der Blöcke von Verzeichnissen und Dateien. Der *Master* beinhaltet außerdem noch das *JournalLog*, in dem aktuelle Dateioperationen gespeichert werden. Auf der Grundlage von *BlockMap* und *JournalLog*, kann der *Master* Anfragen nach Dateien beantworten. Die *DataNodes* sind die *Slave*-Komponenten. Diese sind für die Datenspeicherung zuständig. Soll der *HDFS*-Cluster vergrößert werden, werden *DataNodes* hinzugefügt. Der *NameNode* wird dabei vom *SecondaryNameNode* unterstützt. Der *SecondaryNameNode* entlastet den *NameNode* beim Zusammenführen und Verarbeiten der *BlockMap* und *JournalLog*. (vgl. [Sel15b])

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. [Wik15c])

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-Paaren eine neue Liste von Schlüssel-Wert-Paaren erzeugt. 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 und Abbildung 2 zeigt eine vollständige *MapReduce*-Sequenz.

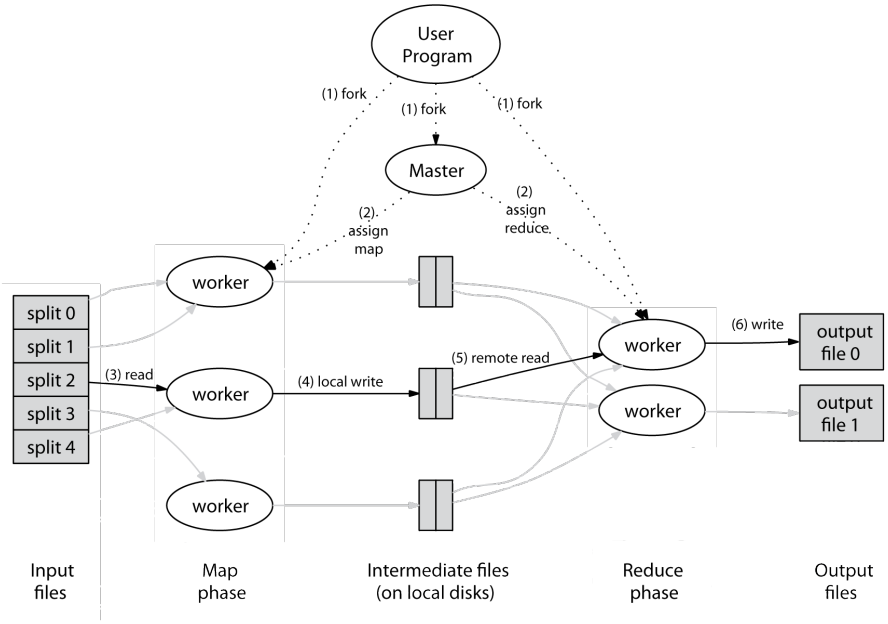

Abbildung 2: Ausführungsreihenfolge von *MapReduce*[DG04]

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 spei-

chert sie temporär 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 Datenaustauschs 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.

3. Fazit

Zusammenfassung
Hadoop ist ein Framework, das konzipiert wurde um große Datenmengen schnell und effizient verarbeiten zu können. Das Framework besteht aus vielen Komponenten, die bestimmte Funktionen übernehmen. So können intensive Rechenprozesse mit großen Datenmengen von großen Computerclustern bearbeitet werden. Ein besonderer Bestandteil von Hadoop ist der *MapReduce*-Algorithmus und das Hadoop Distributed File System. Diese beiden Basis-Komponenten unterstützen Hadoop bei seinen Aufgaben.

Ausblick
Das Thema ist aktuell. Große Firmen wie Facebook, AOL oder IBM verwenden Hadoop. Da es plattformunabhängig ist, kann es in vielen Szenarien verwendet werden. Die Stärken spielt Hadoop aus, wenn große Datenmengen verarbeitet werden müssen. Hadoop wird besonders interessant im Kontext von Big Data. Unternehmen die große Datenmengen analysieren wollen, können Hadoop verwenden. Ein Unternehmen mit relativ wenig Datenmengen, wird nicht durch einen Einsatz von Hadoop profitieren.

A. Anhang

```
1  import com.mongodb.*;
2
3  public class MongoClient {
4    public static void main(String[] args) {
5      Mongo mongo;
6      try {
7        mongo = new Mongo("localhost", 27017);
8        DB db = mongo.getDB("library");
9        DBCollection books = db.getCollection("books");
10
11       BasicDBObject book = new BasicDBObject();
12       book.put("name", "Understanding JAVA");book.put("pages", 100);
13       books.insert(book);
14
15       book = new BasicDBObject();
16       book.put("name", "Understanding JSON");book.put("pages", 200);
17       books.insert(book);
18
19       String map = "function() { "+
20           "var category; " +
21           "if (this.pages >= 250) category = 'Big Books'; " +
22           "else category = 'Small Books'; "+
23           "emit(category, {name: this.name});}";
24
25       String reduce = "function(key, values) { " +
26           "var sum = 0; " +
27           "values.forEach(function(doc) {sum += 1;}); " +
28           "return {books: sum};} ";
29
30       MapReduceCommand cmd = new MapReduceCommand(books, map, reduce,
31       null, MapReduceCommand.OutputType.INLINE, null);
32
33       MapReduceOutput out = books.mapReduce(cmd);
34
35       for (DBObject o : out.results()) {
36           System.out.println(o.toString());
37       }
38     } catch (Exception e) {e.printStackTrace();}
39   }
40 }
```

Listing 1: MapReduce in Java

Literatur

[anG15] av-news GmbH. *Wie Hadoop einsetzen.* 2015. Aufgerufen am: 17.08.2015.

[Bio10] BMC Bioinformatics. *An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics.* 2010. Aufgerufen am: 17.08.2015.

[Bor15] Dhruba Borthakur. *HDFS Architecture Guide.* 2015. Aufgerufen am: 17.08.2015.

[DG04] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters.* OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004. Aufgerufen am: 17.08.2015.

[Fou15] The Apache Software Foundation. *Apache Pig.* 2015. Aufgerufen am: 17.08.2015.

[Inc15] Yahoo! Inc. *Module 2: The Hadoop Distributed File System.* 2015. Aufgerufen am: 17.08.2015.

[Sel15a] Uwe Seller. *Einfuehrung in Hadoop - Die wichtigsten Komponenten von Hadoop.* 2015. Aufgerufen am: 17.08.2015.

[Sel15b] Uwe Seller. *Einfuehrung in Hadoop - Die wichtigsten Komponenten von Hadoop.* 2015. Aufgerufen am: 17.08.2015.

[Wik15a] Wikipedia. *Apache Hadoop.* 2015. Aufgerufen am: 14.08.2015.

[Wik15b] Wikipedia. *Apache Hadoop.* 2015. Aufgerufen am: 17.08.2015.

[Wik15c] Wikipedia. *MapReduce.* 2015. Aufgerufen am: 18.08.2015.

[Wik15d] Wikipedia. *Verteiltes Dateisystem.* 2015. Aufgerufen am: 17.08.2015.

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 23. August 2015 Nils Lutz; Janick Asmuß