Erläuterung von Systemen zur Datenbearbeitung in der Cloud anhand von Apache Hadoop

Tobias Neef

Fakultät für Informatik
Hochschule Mannheim
Paul-Wittsack-Straße 10
68163 Mannheim
tobnee@gmail.com

Zusammenfassung Wachsende Datenmengen und eine nie zuvor dagewesene Menge an potenziellen Endnutzern stellen viele Firmen vor bisher unbekannte Probleme. Wie können Applikationen entwickelt werden, welche diesen Anforderungen gerecht werden? In diesem Artikel werden aktuelle wissenschaftliche Errungenschaften in diesem Gebiet dokumentiert und gezeigt, dass konkrete Implementierungen wie Apache Hadoop, Firmen helfen können ihre IT-Anforderungen umzusetzen und den Weg in das Cloud-Zeitalter zu ebnen.

Seit der Einführung des Internets wachsen Datenberge, die von unseren Softwaresystemen zusammengetragen werden. Untersuchungen zeigen, dass das Wachstum des Datentransfers im Internet ähnlichen Gesetzmäßigkeiten folgt wie denen im Bereich der der Mikroprozessoren [1]. Nicht nur die Datenmenge ist ein Problem; vor allem wollen wir Daten immer stärker zueinander in Verbindung bringen, um interessante Konzepte wie Socal Graphs oder globale Suchindizes umsetzen zu können. Traditionelle Verfahren zur Datenanalyse - beispielsweise über relationale Datenbanken - stoßen in solchen Szenarien an ihre Grenzen. Seit Jahren mussten sich Firmen wie Google oder Facebook damit beschäftigen, mit dieser Problematik umzugehen. Die damit verbundene Forschung hat zu Konzepten geführt, welche es uns erlauben, große Datenmengen über eine Vielzahl von Rechnern zu verteilen und diese effizient zu analysieren. Mittlerweile sind auf Basis dieser Forschungsergebnisse auch OpenSource Systeme entstanden, die es kleineren Firmen erlauben sollen, eine ähnliche Infrastruktur aufzubauen. Das Ziel dieser Arbeit ist, einen Überblick zu schaffen, welche Konzepte heute für die Analyse großer Datenmengen genutzt werden können. Die grundlegenden Konzepte werden im Rahmen dieser Arbeit erläutert und anhand von Beispielszenarien näher gebracht, welche mit Komponenten aus dem Apache Hadoop Projekt umgesetzt sind. Hadoop baut auf den aktuellen Entwicklungen im Bereich verteilter Datenanalyse auf, wodurch es die geeignete Referenz für diese Arbeit darstellt.

1 Aktuelle Entwicklungen in der Datenverarbeitung

Nach heutigem Standard werden Systeme meist auf Basis von relationalen Datenbanken [2] entwickelt, welche in der Praxis schon seit den 70er Jahren im Einsatz sind. Dieser Datenbanktyp arbeitet auf Basis von Entitäten, welche miteinander in Verbindung stehen. Um Informationen aus diesen Enitäten zu gewinnen werden anhand von mathematischen Verknüpfungen dynamische Ergebnismengen gebildet. Dieser Prozess trägt dazu bei, dass relationale Datenbanken eine hohe Datenkonsistenz bieten indem Redundanzen vermieden werden. Des weiteren können komplexe Verknüpfungen in diesem System ausgedrückt werden. Diese und andere Vorteile führen dazu, dass relationale Datenbanken bis in die heutige Zeit die Basis vieler Entwicklungen darstellen. Durch die Etablierung des Internets als allgegenwärtige Kommunikationsplattform sind die globalen Datenmengen deutlich angestiegen. Eine einzelne Applikation verwaltet heute daher oft Datenmengen von hundertern von Gigabyte. In diesen Fällen ist die Performance sowie die Skalierbarkeit der Anwendung ein vorrangiges Designziel, was dazu führt, dass Aspekte wie auftretende Redundanzen in Kauf genommen oder sogar benötigt werden. Die Forschung im Umfeld von großen Internetfirmen hat ergeben, dass relationale Datenbanksysteme diesen Anforderungen nicht genügen, da diese an die Grenzen ihrer Skalierbarkeit stießen und alternative Konzepte benötigt werden um Applikationen dieser Art zu ermöglichen. Daher wurden Forschungsprojekte angestoßen, die großen Einfluss auf die Art und Weise haben wir wir die verteilte Datenverarbeitung heute betrachten. In Folgenden werden die Annahmen erläutert, welche im Design dieser Lösungen als vorrangig identifiziert wurden, sowie konkrete Techniken vorgestellt, welche sich in der Praxis als tauglich erwiesen haben.

2 Annahmen beim Design verteilter Systeme

Beim Entwurf moderner Datenverarbeitungssysteme werden immer diverse Annahmen getroffen, welche die Designentscheidungen leiten [4] [6]. Diese Annahmen basieren auf Erfahrungen, welche im jahrelangen Betrieb solcher Anwendungen gesammelt wurden. Im Hinblick auf den aktuellen Forschungsstand sind folgende Annahmen essentiell:

- **Hardwarefehler sind die Regel:** In einem Netzwerk mit mehreren hundert Systemen ist der Ausfall von Hardware die Regel. Vor allem gilt dies, wenn es sich bei der Hardware um sogenannte Commodity-Hardware handelt, welche keine Ausfallschutzmaßnahmen wie redundante Komponenten vorsehen.
- **Große Dateien sind die primäre Berechnungsbasis:** Da die Systeme auf die Berechnung von großen Datenmengen optimiert werden, ist der Ausgangspunkt für die meisten Berechnungen eine oder mehrere große Dateien.

– **Optimierung auf lesenden Dateizugriff:** Bei den Eingangs erwähnten Szenarien (Suchindizes, Socal Graphs) geht man davon aus, dass der Regelfall der Datennutzung ein einmal-geschrieben-mehrfach-gelesen Muster ist.
– **Netzwerkbandbreite ist der limitierende Faktor:** In einem großen Netzwerk von Systemen kann man relativ einfach durch das Hinzufügen neuer Systeme die Speicher und Rechenleistung des Gesamtsystems erhöhen. Der limitierende Faktor ist die Netzwerkbandbreite, welche nicht so einfach durch Hinzufügen von zusätzlicher Hardware skaliert.

3 Techniken für die verteilte Datenverarbeitung

Im Folgenden werden zwei Techniken vorgestellt, welche die Basis vieler moderner Systeme darstellen und auf der Forschungsseite bereits ausführlich ergründet wurden. Bei diesen Techniken handelt es sich um das Programmiermodell MapReduce [6] sowie eine Datenverteilungsinfrastruktur in Form eines verteilten Dateisystems [4]. In Kombination entwickeln diese Techniken starke Synergieeffekte, welche die Relevanz für die verteilte Datenverarbeitung nochmals hervorheben.

3.1 Das Programmiermodell MapReduce

Bei MapReduce handelt es sich um ein Programmiermodell, welches für die Verarbeitung sehr großer Datenmengen geeignet ist. Dies geschieht durch die inhärente Parallelität des Modells, welches bewusst einfach gehalten ist, ohne jedoch zu einfach zu sein um sinnvolle Problematiken damit lösen zu können. Wie durch den Namen bereits kenntlich gemacht, spezifiziert der Benutzer eine Map-Funktion, welche ein Schlüssel-Wertepaar akzeptiert und daraus eine von temporären Schlüssel-Wertepaaren erzeugt. Zusätzlich wird eine Reduce-Funktion spezifiziert, die alle Werte entgegen nimmt, welche den gleichen Schlüssel besitzen, um diese zu einem Ergebnis weiter zu verarbeiten. Die Verteilung der Berechnungen sowie die Zuordnung der Ergebnissen der Map zu den Reduce Funktionen wird vom Modell abstrahiert.

3.2 Beispiel einer Map und Reduce Funktion

Ein Beispiel, welches die Funktionsweise von MapReduce zeigt, ist das Zählen von Besuchern einer URL anhand der Logdateien von Webservern.

```
map (String schluessel, String wert):
  // schluessel: Der Name der Logdatei
  // wert: Der Inhalt der Logdatei
  foreach line z in wert:
    // extrahiert die URL aus einer Zeile im Log
```

```
    url = getURL(z)
    // füge den Tupel (URL,1) zur Ergebnismenge der Funktion hinzu
    sammle(url,1)

reduce (String schluessel, Iterator werte)
  // schluessel: Eine URL
  // wert: In diesem Fall immer 1
  foreach value v in werte:
    anzahl += v
  sammle(anzahl)
```

Listing 1.1. MapReduce Beispiel

Die Ablaufumgebung könnte nun für jedes Log eine Map-Funktion aufrufen. Diese würden dann für jede aufgerufene URL ein Paar (url,1) sammeln.

```
(www.beispiel.de/impressum, 1)
(www.beispiel.de/impressum, 1)
(www.beispiel.de/meet, 1)
```

Anschließend würden alle Werte mit dem gleichen Schlüssel der Reduce-Funktion übergeben, die die Werte aufsummiert und so die Gesamtzahl der Einträge pro Schlüssel/URL berechnet.

```
(www.beispiel.de/impressum, 2)
(www.beispiel.de/meet, 1)
```

3.3 Datenverteilungsinfrastruktur

Um den Entwickler eines MapReduce Jobs die Möglichkeit zu geben, sich auf die Definition seines Algorithmus zu Konzentrieren besteht der Bedarf für eine Infrastruktur, welche diverse im Annahmekapitel beschriebene Rahmenbedingungen abstrahiert. Google beschreibt in dem Artikel zum Google File System eine solche Infrastruktur. Im folgenden Kapitel wird eine OpenSource Implementierung dieser Konzepte im Detail vorgestellt.

4 Überblick Apache Hadoop

4.1 Geschichte

Hadoop startete unter den Namen Nutch als Subprojekt von Apache Lucene, einer erfolgreichen Textsuchengine. Das Ziel war eine Websearchengine zur Entwickeln, welche es ermöglichen sollte das Web zu durchsuchen und zu indexieren und dabei die Wartungsaufwände minimal zu halten. In 2002 konnte eine erste Version entwickelt werden, bei der sich jedoch herausstellte, dass dieser Ansatz

nicht für Milliarden von Seiten skalieren würde. In den Jahren 2003 und 2004 veröffentliche Google die Paper zu GFS und MapReduce, welche die weitere technische Richtung für das Projekt vorgaben [7]. Mittlerweile ist Hadoop ein eigenständiges TopLevel Apache-Projekt mit diversen eigenen Subprojekten und im Einsatz bei diversen Organisationen wie Facebook oder Yahoo.

4.2 Projekte

Die Hadoop Projekte lehnen sich größtenteils an die Veröffentlichungen von Google an, wobei auch zunehmend Applikationen entstehen, die Hadoop als Basisinfrastruktur nutzen, um Dienste wie verteilte Datenbanken oder DataWarehouses bereitstellen zu können. Implementiert sind die Hadoop-Projekte in der Programmiersprache Java.

- **HDFS (Hadoop Filesystem):** Ist das verteile Dateisystem, welches auf Basis der GFS Veröffentlichung von Google entwickelt wurde.
- **MapReduce:** Das Projekt beschäftigt sich mit dem Programmiermodell für die Verteilte Dateverarbeitung.
- **ZooKeeper:** Hierbei handelt es sich um ein Koordinationsservice für verteilte Anwendungen.
- **Pig:** Ist eine Datenflusssprache, welche auf Basis von MapReduce/HDFS läuft [5].
- **HBase:** Hierbei handelt es sich um eine verteilte spaltenorientierte Datenbank, welche HDFS als Speichersystem nutzt sowie ZooKeeper als Lockingsystem. Datenbankabfragen werden anhand von MapReduce für die Batchverarbeitung oder anhand von PointSeeks für Zugriffszeit optimierte Abfragen umgesetzt.

5 Architektur und Konzepte vom HDFS

Im Folgenden sollen die Kernaspekte der Hadoop-Filesystem Architektur erläutert werden, die auch in Abbildung 1 schematisch dargestellt sind.

5.1 Blocks

Blocks stellen in Speichersystemen meist die kleinste Einheit da, die adressiert werden kann. In Festplatten sind Blocks normalerweise 512 Byte groß, in Dateisystemen meist ein vielfaches davon. Blocks in einem Netzwerkdateisystem wie HDFS liegen im Abstraktionslevel über diesen Komponenten um dem Benutzer zu ermöglichen eine Ansammlung von Maschinen als eine Speichereinheit zu sehen. Da HDFS auf das Lesen von großen Dateien optimiert ist sind typische Blocks in HDFS 64MB oder größer. Die Wahl einer großen Blockgröße beeinflusst auch das Verhältnis von Such- zu Transferzeit. Bei kleineren Blöcken muss immer

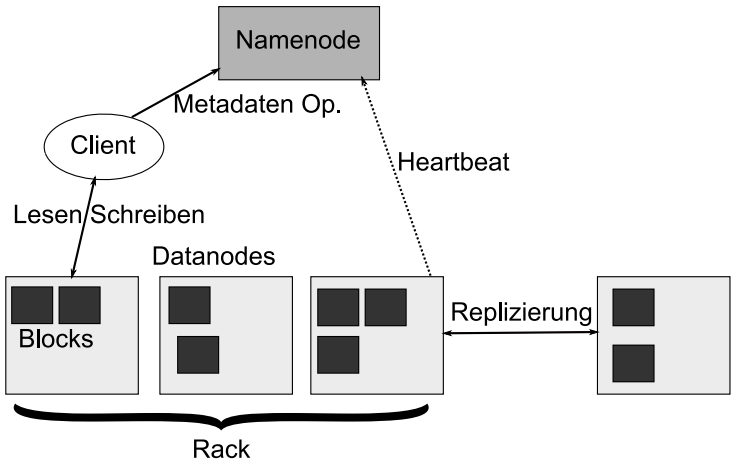

Abbildung 1. HDFS Architektur nach [3]

wieder über das Netzwerk der Folgeblock ausfindig gemacht werden, was dazu führen kann das die Suchzeit erheblich zur Gesamtübertragungszeit beiträgt. Die Vorteile der Blockabstraktion sind vielfältig: So können beliebig große Dateien in HDFS gespeichert werden, solang die Summe aller im Cluster verfügbaren Speicher dies nicht limitiert. Einzelne Dateien werden also nicht auf einem PC gespeichert, sondern werden über das Netzwerk verteilt. Dies vereinfacht Aspekte wie Verfügbarkeit und Fehlertoleranz auf einem hohen Niveau anbieten zu können, indem man Blocke nicht nur einfach über das Netzwerk verteilt, sondern mehrere Kopien vorhält. Standardmäßig werden in HDFS drei Kopien eines Blocks im System gehalten. Damit können auch lokale Sicherheitsvorrichtungen wie Raid-Systeme überflüssig gemacht werden. Bei einigen gefragten Dateien kann es auch sinnvoll sein die Replikationsrate zu erhöhen um damit im Netzwerk mehr Knoten zu haben die den selben Block bereitstellen.

5.2 Nodes

HDFS setzt auf zwei Typen von Knoten, welche innerhalb des Gesamtsystems unterschieden werden. Dabei handelt es sich um den Namenode und die Datanodes. Der Namenode existiert logisch nur ein mal in einem HDFS-Cluster. Er ist verantwortlich für die Verwaltung von Metainformationen wie Dateinamen oder Berechtigungen. Des Weiteren weiß der Namenode über die Lage der Blocks eines Files Bescheid. Der Namenode ist aber nicht für die eigentliche Datenübertragung verantwortlich da diese über die Datanodes geht. Wenn ein HDFS-Client auf eine Datei zugreifen will, interagiert er also immer mit beiden Knotentypen auch wenn dies für den Benutzer transparent geschieht. Die Datanodes melden sich in einem regelmäßigen Rhythmus (Heartbeat) bei ihrem

Namenode mit Informationen über die gespeicherten Blöcke. Ohne den Namenode wäre das gesamte Filesystem nicht mehr nutzbar, da es keine Möglichkeit gäbe die Zuordnung von Dateien zu Blöcken wiederherzustellen. Damit ist der Namenode eine kritische Komponente in einem HDFS-System und wird daher entgegen der allgemeinen Architektur zumeist auf einer ausfallsicheren Hardware betrieben. Es bestehen zudem Möglichkeiten Schattensysteme bereitzustellen, welche die Ausfallsicherheit erhöhen sowie den primären Namenode entlasten, indem Lesezugriffe auf die Schattensysteme verteilt werden.

5.3 Interfaces

Es gibt diverse Möglichkeiten mit HDFS zu interagieren. Eine einfache Möglichkeit ist das CLI, welches für diverse administrative Aufgaben gut geeignet ist. Da Hadoop in Java geschrieben ist, werden alle Serviceaufrufe an das Filesystem über die Java API geleitet. Mit JVM-Sprachen kann so direkt auf die Filesystem-API zugegriffen werden. Für C gibt es ebenfalls eine Bibliothek, welche mit Hilfe des Java Native Interfaces realisiert ist, aber funktional der Java Implementierung nachsteht. Da HDFS nicht voll POSIX kompatibel ist, kann es auch keine nativen Support im Kernel geben. Um trotzdem eine ähnliche Einbindung zu ermöglichen kann ein FUSE kompatibler Treiber im Benutzerkontext geladen werden, welcher es ermöglicht Hadoop Dateisysteme in einem Unix System zu mounten. Alternativ ist es noch Betriebsystemübergreifend möglich über eine WebDAV-Schnittstelle auf HDFS zuzugreifen.

6 Hadoop und die Cloud

Der Zusammenhang zwischen Cloud Computing und Hadoop ist auf den ersten Blick nicht unbedingt gegeben. Bei genauerem hinsehen erweisen sich diese beiden Konzepte als perfekte Ergänzung. Beim Arbeiten mit der Cloud soll es dem Nutzer ermöglicht werden skalierbare Applikationen anbieten zu können ohne sich selbst um die Wartung einer solchen Infrastruktur kümmern zu müssen. Ein häufiger Irrtum hingegen ist, dass die Applikation durch das reine Deployment in der Cloud automatisch skaliert und nur zusätzliche Rechenleistung bereitgestellt werden muss um die Geschwindigkeit zu erhöhen. Bei den heutigen Infrastructure As A Service Anbietern kann man sich Rechenkapazität in Form von virtuellen Computer kaufen, welche aber nur bis zu einem definierten Stand ausgebaut werden können. Ist die beste Konfiguration gewählt, müssen zur Performanzsteigerung zusätzliche virtuelle Computer gekauft werden. Dies hat zur Konsequnz, dass darauf laufende Applikationen auf zumindest zwei Knoten lauffähig sein müssen. Idealerweise sollte die Applikation auf beliebig vielen Knoten laufen können und dabei mit der Anzahl der Knoten skalieren. Diese Aufgabe liegt in der Hand des Nutzers eines IaaS Anbieters und Projekte wie Hadoop können dabei helfen dieser Herausforderung gerecht zu werden. Beispielsweise nutzte die New York Times die Kombination aus Amazon EC2/Hadoop, um 4TB-Artikel

in PDF zu archivieren. Auf 100 Nodes konnte die Aufgabe in etwa 100 Stunden gelöst werden, was zu einer deutlichen Kosteneinsparung führte [9]. Unter anderem motiviert durch solche Erfolgserlebnisse offerieren die großen Clouddienstleister Plattformen, welche solche Konzepte bereits eingebaut haben. Beispielsweise kann man bei Amazon mittlerweile den Amazon Elastic MapReduce Dienst nutzen, der auf Hadoop MapReduce aufbaut aber dem Benutzer die manuelle Installation und Konfiguration auf EC2-Instanzen erspart.

7 Darstellung eines Beispielszenarios

Um die in den vorhergehenden Kapiteln beschriebenen Konzepte näher zubringen, wird in diesem Kapitel eine Hadoop Beispielanwendung dargestellt. Ziel der Anwendung ist es, die Wörter in den gesammelten Werken Shakespeares über einen invertierten Index der entsprechenden Position in seinen Büchern zuordnen zu können. Das Resultat soll also eine Datei nach diesem Schema sein.

```
thee     lucrece@offset, phoenix@offset, ...
money    venus@offset, ...
```

Um die Daten für den MapReduce Job bereitzustellen müssen diese über HDFS verfügbar sein. Hierzu laden wir die Dateien über die HDFS-Console vom lokalen Dateisystem in HDFS (Ordner input unterhalb des Root-Verzeichnises). Um den Job zu konfigurieren, wird in Hadoop eine Jobkonfiguration in Form einer Klassenimplementierung anlegt. In unserem Fall verweisen wir auf die entsprechenden Verzeichnisse für Ein und Ausgabe sowie die zu definierende Map und Reduce Klasse.

```
public class LineIndexer extends Configured implements Tool {

  private static final String OUTPUT_PATH = "output";
  private static final String INPUT_PATH = "input";

  private void runJob() throws IOException {

    JobConf conf = new JobConf(getConf(), LineIndexer.class);

    // Verzeichnisse
    FileInputFormat.addInputPath(conf, new Path(INPUT_PATH));
    FileOutputFormat.setOutputPath(conf, new Path(OUTPUT_PATH));

    // Map und Reduce Klassen
    conf.setMapperClass(LineIndexMapper.class);
    conf.setReducerClass(LineIndexReducer.class);

    // Datentyp von Key / Value
    conf.setOutputKeyClass(Text.class);
```

```
    conf.setOutputValueClass(Text.class);

    JobClient.runJob(conf);
  }

  public int run(String [] args) throws IOException {
    runJob();
    return 0;
  }

  public static void main(String [] args) throws Exception {
    int ret = ToolRunner.run(new LineIndexer(), args);
    System.exit(ret);
  }
}
```

Listing 1.2. Hadoop Job-Konfiguration

Die Mapper-Klasse wird nun aufgerufen für jede Zeile eines Buches. Als Schlüssel wird der Zeilenoffset angegeben. Der Inhalt der Zeile ist der Wert. Zuerst extrahieren wir den Dateinamen aus der Jobbeschreibung und anschließend iterieren wir über alle Wörter der Reihe. Für jedes Wort wird das Tupel (key: Wort, value: filename@werk) für die weitere Verarbeitung bereitgestellt.

```
public class LineIndexMapper extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, Text> {

  public LineIndexMapper() { }

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, Text> output,
                  Reporter reporter) throws IOException {
          FileSplit fileSplit = (FileSplit)reporter.getInputSplit();
          String fileName = fileSplit.getPath().getName();
          Scanner sc = new Scanner(value.toString());
          while(sc.hasNext()) {
                  Text oKey = new Text(sc.next());
                  Text oValue = new Text(fileName+'@'+key);
                  output.collect(oKey, oValue);
          }
  }
}
```

Listing 1.3. Hadoop Map Funktion

Die Instanzen der Reducer-Klasse können nun mit einem Wort als Schlüssel sowie einem Iterator über alle zugeordneten Vorkommnisse durchlaufen werden. Um

an das gewünschte Ergebnis zu kommen, müssen wir also lediglich alle Einträge des Iterators zu einem Ergebnisstring konkatenieren.

```
public class LineIndexReducer extends MapReduceBase
    implements Reducer<Text, Text, Text, Text> {

  public LineIndexReducer() { }

  public void reduce(Text key, Iterator<Text> values,
      OutputCollector<Text, Text> output, Reporter reporter)
        throws IOException {
          StringBuffer sb = new StringBuffer();
          while(values.hasNext()) {
                  Text value = values.next();
                  sb.append(value.toString());
                  if(values.hasNext()) sb.append(',');
          }
          output.collect(key, new Text(sb.toString()));
  }
}
```

Listing 1.4. Hadoop Reduce-Funktion

Im Ergebnis steht jetzt ein invertierter Index zu Verfügung, welcher beispielsweise in eine Datenbank wie HBase (siehe Hadoop-Projekte) importiert werden kann um so für Suchanfragen innerhalb eines Dialogsystems optimiert zu sein. Durch die automatische Verteilung der Daten wäre es möglich diese Berechnung auf einer Vielzahl von Knoten durchzuführen um somit mit steigenden Textgrößen skalieren zu können.

8 Zusammenfassung

Lösungen wie Hadoop ermöglichen es heute Firmen jeder Größe mit großen Datenmengen umgehen zu können. Diese Vorteile haben jedoch den Preis, dass Teile der Applikation in MapReduce formuliert werden müssen, was für die meisten Entwickler eine Umstellung ist. Durch die vielfältigen Integrationsmöglichkeiten von Hadoop werden diese Einstiegshürden jedoch niedrig gehalten und der Erfolg bei Unternehmen wie Yahoo oder der New York Times zeigt, dass dieser Mehraufwand durchaus berechtigt ist.

Literatur

1. K.G. Coffman, Andrew Odlyzko. *Handbook of Massive Data Sets*. Kluwer Academic. p. 47-93. 2001

2. EF Codd *A relational model of data for large shared data banks*. Communications of the ACM (13) 6, p. 377-387. Juni 1970
3. D Borthakur *The hadoop distributed file system: Architecture and design*. Hadoop Project Website - hadoop.apache.org. 2007
`http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf`
4. S Ghemawat, H Gobioff, ST Leung. *The Google file system*. ACM SIGOPS Operating Systems Review (37) 5, p. 29 - 43, Dezember 2003
5. Christopher Olston et al. *Pig latin: a not-so-foreign language for data processing*. Proceedings of the 2008 ACM SIGMOD international conference on Management of data (1) 2, p. 1099-1110. August 2008
6. J Dean, S Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. 6th Symposium on Operating System Design and Implementation p. 137-150. Dezember 2004
7. Tom White et al. *Hadoop: The Definitive Guide*. OReilly Media. Juni 2009
8. Jason Venner *Pro Hadoop*. Apress; 1 edition. Juni 2009
9. Derek Gottfrid *Self-service, Prorated Super Computing Fun*. November 2007
`http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun`