Institut für Parallele und Verteilte Systeme

Abteilung Anwendersoftware

Universität Stuttgart
Universitätsstraße 38
D – 70569 Stuttgart

Fachstudie Nr. 148

Fachstudie MapReduce - Eine vergleichende Analyse aktueller Implementierungen

Stanislaus Biesinger, Michael Pitterle, Mert Canko

Studiengang: Softwaretechnik

Prüfer: PD Dr. rer. nat. Holger Schwarz
Betreuer: PD Dr. rer. nat. Holger Schwarz
begonnen am: 01.12.2011
beendet am: 20.07.2012

CR-Klassifikation: C.2.4, H.2, H.2.3

Inhaltsverzeichnis

1 Einleitung 1

1.1 Was ist MapReduce....1

1.1.1 Funktionsweise....1

1.1.2 Beispiel....2

1.1.3 Nutzen....2

1.2 Inhalt der Fachstudie....2

1.3 Begriffsklärung....3

2 Vorauswahl relevanter MapReduce-Implementierungen 4

2.1 Entwicklungsaktivität....4

2.2 Alleinstellungsmerkmale....4

2.3 Lokale Testbarkeit....5

2.4 Geschätzte Einarbeitungszeit....5

2.5 Existenz einer Nutzerbasis....5

2.6 Medienpräsenz/Popularität....6

2.7 Ergebnis der Vorauswahl....7

3 Bewertung der MapReduce-Implementierungen 8

3.1 Bewertungsschema für den ersten Vergleichstest....8

3.2 Hadoop-basierende MapReduce-Implementierungen....10

3.2.1 Apache Hadoop....10

3.2.2 MapR....11

3.2.3 Cloudera....12

3.2.4 Cascading....13

3.3 Nicht auf Hadoop basierende Implementierungen von MapReduce....14

3.3.1 Basho Riak....14

3.3.2 Aster Data SQL-MapReduce....15

3.3.3 MongoDB....16

3.3.4 Nokia Disco....17

3.3.5 Spark....18

3.3.6 Myspace Qizmt....18

3.4 Implementierungen mit MapReduce ähnlichen Programmiermodellen....19

3.4.1 Gearman....19

3.4.2 Stratosphere....20

3.4.3 GridGain....21

3.4.4 Sector/Sphere....21

3.5 Übersicht der Bewertungsergebnisse....23

3.6 Die näher zu betrachtenden 6 Kandidaten....23

4 Detaillierte Betrachtung ausgewählter Implementierungen 26

4.1 Kriterien und Parameter für die detaillierte Betrachtung....26

4.1.1 Kriterien und Vergleichsaspekte....26

4.1.2 Erläuterung der Fragestellung der Vergleichsaspekte....28

4.2 Verwendete Testsysteme....31

4.3 Apache Hadoop....32

4.3.1 Installation, Inbetriebnahme und Dokumentation....33

4.3.2 Erstellung von MapReduce-Funktionen in Java....33

4.3.3 Ausführen von MapReduce-Aufträgen....35

4.3.4 Skalierung....35

4.3.5 Lastbalancierung....36
4.3.6 Dateisystem....36
4.3.7 Fehlertoleranz....36
4.3.8 Monitoring....36
4.3.9 Fazit....38
4.4 MapR....38
4.4.1 Installation, Inbetriebnahme und Dokumentation....38
4.4.2 Erstellung von MapReduce-Funktionen....39
4.4.3 Ausführen von MapReduce-Aufträgen....39
4.4.4 Skalierung....40
4.4.5 Lastbalancierung....40
4.4.6 Dateisystem....40
4.4.7 Fehlertoleranz....42
4.4.8 Monitoring....42
4.4.9 Sonstiges....45
4.4.10 Fazit....47
4.5 GridGain....48
4.5.1 Installation, Inbetriebnahme und Dokumentation....48
4.5.2 Erstellung von MapReduce-Funktionen in Java....48
4.5.3 Ausführen von MapReduce-Aufträgen....50
4.5.4 Skalierung....50
4.5.5 Dateisystem....50
4.5.6 Fehlertoleranz....51
4.5.7 Monitoring....51
4.5.8 Fazit....52
4.6 Basho Riak....52
4.6.1 Installation, Inbetriebnahme und Dokumentation....53
4.6.2 Erstellung von MapReduce-Funktionen....54
4.6.3 Ausführen von MapReduce-Aufträgen....55
4.6.4 Skalierung....55
4.6.5 Lastbalancierung....56
4.6.6 Dateisystem....56
4.6.7 Datenkonsistenz und Fehlertoleranz....57
4.6.8 Monitoring....57
4.6.9 Sonstiges....59
4.6.10 Fazit....61
4.7 MongoDB....62
4.7.1 Installation, Inbetriebnahme und Dokumentation....62
4.7.2 Erstellung von MapReduce Funktionen....62
4.7.3 Ausführen von MapReduce-Aufträgen....63
4.7.4 Skalierung....63
4.7.5 Lastbalancierung....64
4.7.6 Dateisystem....64
4.7.7 Fehlertoleranz....64
4.7.8 Monitoring....64
4.7.9 Fazit....65
4.8 MySpace Qizmt....65
4.8.1 Installation, Inbetriebnahme und Dokumentation....65
4.8.2 Erstellung von MapReduce-Funktionen....65
4.8.3 Ausführen von MapReduce-Aufträgen....68
4.8.4 Skalierung....68
4.8.5 Dateisystem....68
4.8.6 Fehlertoleranz....69
4.8.7 Monitoring....69
4.8.8 Fazit....69
4.9 Übersicht über die Ergebnisse der detaillierten Betrachtung....70

5 Bewertung der Ergebnisse **71**
5.1 Allgemein....71
5.2 Genau betrachtete Implementierungen....72
5.3 Abschließende Worte....73

Literaturverzeichnis **75**

Anhang **77**
Anhang 1.1 Übersicht über mögliche Implementierungen
Anhang 1.2 Daten-Tabelle der Implementierungen
Anhang 1.3 KO-Tabelle für die Auswahl der Implementierungen
Anhang 2.1 Ergebnisse der Punktebewertung
Anhang 2.2 Überblick über die Bewertungsergebnisse
Anhang 2.3 Vorauswahl Bewertungsergebnisse im Detail
Anhang 3.1 Übersicht über die Ergebnisse der detaillierten Betrachtung

1 Einleitung

1.1 Was ist MapReduce

MapReduce ist ein im Jahr 2004 von Google vorgestelltes Framework für das Bearbeiten hoch parallelisierbarer Aufgaben auf großen Datenmengen. Es nutzt hierzu eine große Anzahl von einzelnen Rechnern (Knoten), welche kollektiv als Cluster bezeichnet werden (beziehungsweise als "Grid", falls sich die Hardware der Knoten unterscheidet) [DNS06].

1.1.1 Funktionsweise

MapReduce besteht aus den namensgebenden Schritten "Map" und "Reduce", welche in ihrer Funktionalität auf den gleichnamigen Funktionen vieler funktionaler Programmiersprachen basieren. Die den Schritten zu Grunde liegenden logischen Funktionen unterscheiden sich je nach zu implementierender Berechnungsaufgabe. Einzige Konstanten sind die jeweilige Ein- und Ausgabe. Die Eingaben sind als (Schlüssel, Wert)-Paar definiert, die Ausgabe ist beim Map-Schritt eine Liste von (Schlüssel, Wert)-Paaren, beim Reduce-Schritt eine Liste von Werten, üblicherweise aus Null oder genau einem Wert bestehend.

Der Map-Schritt

Im Map-Schritt empfängt ein übergeordneter "Masterknoten" die Dateneingabe, teilt diese in kleinere Segmente auf und verteilt sie an die restlichen Knoten (Worker). In einigen Implementierungen (inklusive Googles Original) ist es den Workern möglich, den empfangenen Datensatz ihrerseits nochmals aufzuteilen und an weitere Knoten zu delegieren. Jeder Knoten bearbeitet gemäß der zuvor definierten logischen Funktion seinen Teildatensatz, und gibt diesen an seinen Masterknoten weiter.

Der Reduce-Schritt

Im Reduce-Schritt werden die vom Masterknoten erhaltenen (Schlüssel, Wert)-Paare wieder an die Workerknoten verteilt, welche sie gemäß der Reduce-Funktion zu Listen zusammenfassen. Diese Teillisten werden wiederum vom Masterknoten zu einer endgültigen Liste zusammengefasst, welche die Lösung der ursprünglichen Aufgabe darstellt.

1.1.2 Beispiel

Dieses Beispiel, welches aus der ursprünglichen MapReduce-Veröffentlichung stammt, zählt das Vorkommen eines Wortes w in einem Satz von Dokumenten.

```
1   void map(String name, String document):
2     // name: document name
3     // document: document contents
4     for each word w in document:
5       EmitIntermediate(w, "1");
6
7   void reduce(String word, Iterator partialCounts):
8     // word: a word
9     // partialCounts: a list of aggregated partial counts
10    int sum = 0;
11    for each pc in partialCounts:
12      sum += ParseInt(pc);
13    Emit(word, AsString(sum));
```

Die Map-Funktion gibt hier pro Vorkommen des Wortes w in einem Dokument ein (Schlüssel, Wert)-Paar aus, welches als Schlüssel w und als Wert die Konstante 1 besitzt. Die Reduce-Funktion zählt anschließend die Konstanten zusammen und gibt die Gesamtanzahl aus.

1.1.3 Nutzen

MapReduce erlaubt einen sehr hohen Grad an Parallelisierung, welcher besonders bei sehr großen Datenmengen einen enormen Geschwindigkeitszuwachs bedeutet. Solange jede Map-Berechnung von allen anderen unabhängig ist und genügend Knoten bereitstehen, können sämtliche Berechnungen gleichzeitig durchgeführt werden. Selbiges gilt für die Reduce-Funktion, unter der Voraussetzung, dass sämtliche (Schlüssel, Wert)-Paare mit demselben Schlüssel zeitgleich auf einem Knoten vorliegen.

Darüber hinaus bietet das MapReduce-Framework auch einen gewissen Grad an Ausfallsicherheit, da bei verzögertem Eintreffen von Berechnungsergebnissen die Eingabedaten einem neuen Worker zugewiesen werden können.

1.2 Inhalt der Fachstudie

In dieser Fachstudie werden mehrere MapReduce-Implementierungen untersucht und verglichen. Um die geeigneten Kandidaten für den Vergleich zu finden, ist also eine ausführliche Recherche bezüglich der Funktionalität von MapReduce, sowie die Erfassung der vorhandenen Implementierungen inklusive der vielen MapReduce-ähnlichen Konzepte notwendig.
Die Fachstudie enthält unter anderem eine Sammlung von recherchierten MapReduce Implementierungen. Zu diesen gehörten neben dem Namen, einer Referenz auf die Entwickler-Website, den möglichen Lizensierungsformen und der unterstützten Betriebssysteme auch das Datum des aktuellsten Releases. Inaktive oder fragwürdige Projekte wurden bereits bei Aufnahme in die Liste von der weiteren Betrachtung ausgeschlossen, ihre Aufnahme erfolgte nur der Vollständigkeit halber. Diese Projekte sind in der Übersichtstabelle in *Anhang 1.1* mit einer grauen Hintergrundfarbe markiert. Außerdem wurden auch schon einige einzigartige oder heraus stechende Eigenschaften der verschiedenen Vertreter erkannt und ebenfalls notiert.
Da noch nicht klar war, ob auch MapReduce-ähnliche Implementierungen untersucht würden, und viele Vertreter des MapReduce Paradigmas auf Apache Hadoop basieren, wurde nach Hadoop basierten Distributionen, nicht-Hadoop-Implementierungen sowie MapReduce-ähnlichen Implementierungen sortiert.

Während dieses Prozesses sammelten sich über 40 Kandidaten an, wobei davon etwa ein Viertel als inaktive Projekte markiert wurden (vgl. *Anhang 1.1* sowie *Anhang 1.2*). Auf Basis der verbliebenen 30 Implementierungen musste nun eine engere Auswahl getroffen werden, mit welcher sinnvolle Vergleichskandidaten gefunden werden konnten.
Im Team wurden Möglichkeiten besprochen, diese Menge mithilfe subjektiver Eindrücke, Relevanz bezüglich Lizenzen und Umsetzbarkeit, jedoch auch mithilfe objektiver Kriterien zu verkleinern. Dabei wurden sowohl KO- als auch Auswahlkriterien diskutiert. Einen Überblick über die verbliebenen Kandidaten findet man in *Anhang 1.3*. Die Ergebnisse werden in *Kapitel 2* detailliert vorgestellt. Zudem werden die Kriterien der Vorauswahl, sowie das Vorgehen bei der Suche nach geeigneten Kandidaten erläutert.
Mithilfe der beschlossenen Kriterien konnte die Menge auf 14 Kandidaten eingeschränkt werden, die vom Team selektiert und als für die Fachstudie geeignet eingestuft wurden. Um endgültige Implementierungen für eine tiefer greifende Untersuchung zu finden, haben wir uns darauf geeinigt, die besonderen Merkmale der Kandidaten vorzustellen und mit den anderen Eigenschaften und Features nach eigens erhobenen Kriterien zu bewerten. In *Kapitel 3* werden die Kandidaten aufgelistet so wie das Bewertungsschema vorgestellt. Im darauffolgenden *Kapitel 4* werden die Kandidaten vorgestellt sowie die Bewertungsergebnisse vorgestellt und erläutert. Die dabei verwendeten Quelltexte stammen aus den Beispielen, welche vom Hersteller zu Verfügung gestellt wurden. Diese wurden ggf. noch angepasst oder gekürzt. In *Kapitel 5* wird das Gesamtergebnis der Fachstudie betrachtet.

1.3 Begriffsklärung

In dieser Fachstudie wird ein gewisses Maß an fachspezifischer Terminologie verwendet. Einige der Konzepte und Begrifflichkeiten sind dem Leser eventuell nicht bekannt. Aus diesem Grund werden in diesem Abschnitt einige dieser Begriffe erläutert. Die MapReduce-spezifischen Terminologien sind [DNS06] entnommen. Das MapReduce-Paradigma basiert auf so genannten *Clustern*. Hiermit ist eine Gruppe von handelsüblichen Computern gemeint welche als Schwarm fungieren, ihnen übergebene Arbeitsaufträge (*Jobs*) also in Gemeinschaft bearbeiten. Die einzelnen Maschinen werden hierbei als *Knoten* bezeichnet. Die *Knoten* sind hierarchisch organisiert: Der *Masterknoten* koordiniert die Verteilung von Arbeitsaufträgen an die restlichen Knoten, welche als *Slaveknoten* bezeichnet werden. Falls ein einzelner Knoten durch seinen Ausfall die Funktionsfähigkeit des *Clusters* aufhebt, so wird dieser als *Single Point of Failure* bezeichnet[SPF].
Viele der in diesem Dokument behandelten MapReduce-Implementierungen besitzen ein eigenes verteiltes Dateisystem. Unter diesem Begriff versteht man einen Computercluster, welcher Daten verteilt auf mehreren Maschinen speichern, und diese über einen Satz Metadaten wieder auffinden kann. Sowohl die durch die Ausführung von Jobs entstehende Arbeitslast, sowie die Lese- und Schreiblast auf dem verteilten Dateisystem, können so auf die einzelnen Knoten verteilt werden, dass eine möglichst gleichmäßige Lastverteilung entsteht. Dieses Konzept wird als Lastbalancierung (*Load Balancing)* bezeichnet[LBC]. Der Begriff des *Monitorings* bezeichnet in diesem Dokument die Überwachung des Zustands des Clusters.

2 Vorauswahl relevanter MapReduce-Implementierungen

Zunächst wurde mithilfe von Suchmaschinen, Foreneinträgen und Querverweisen nach existierenden MapReduce-Implementierungen gesucht und deren grundlegenden Eigenschaften in Tabellenform erfasst (vgl. *Anhang 1.1* sowie *Anhang 1.2*). Hierbei wurde festgestellt, dass seit seiner Veröffentlichung im Jahr 2006 eine Vielzahl von Implementierungen des MapReduce-Paradigmas entstanden sind. Um die Masse an Möglichkeiten auf ein bearbeitbares Niveau zu reduzieren, wurden folgende Auswahlkriterien für eine erste Vorauswahl von Implementierungen angewandt:

- Entwicklungsaktivität,
- Alleinstellungsmerkmale,
- Lokale Testbarkeit,
- Geschätzte Einarbeitungszeit,
- Existenz einer Nutzerbasis,
- Medienpräsenz/Popularität.

Diese werden im Folgenden anhand einiger Beispiele kurz erläutert.

2.1 Entwicklungsaktivität

Bezüglich der angestrebten Relevanz der Fachstudie ist es vonnöten, für die Zielsetzung des Publikums interessante Implementierungen des MapReduce-Paradigmas als Vergleichskandidaten zu wählen. Hierfür eignen sich besonders diejenigen Implementierungen, welche fortlaufend entwickelt werden. Die Inaktivität eines Implementierungsprojektes muss jedoch nicht zwangsläufig zum Ausschluss aus dem Vergleich führen, da bei manchen Konzepten nicht nur die konkrete Implementierung, sondern der Ansatz als solcher interessant ist (siehe 2.2 Alleinstellungsmerkmale). Um Informationen über die Entwicklungsaktivität eines Projektes zu erlangen, bedienten wir uns den jeweiligen Projektwebsites. Diese hatten oftmals ein Repository, welchem das Datum des letzten Commits zu entnehmen war. In diesem Falle gingen wir davon aus, dass diejenigen Implementierungen, deren Quellcode innerhalb der letzten drei Monate bearbeitet wurde, als aktiv zu werten sind. In denjenigen Fällen, in denen der Quelltext nicht einsehbar, beziehungsweise das letzte Änderungsdatum nicht erkennbar war, wurde die Website des Projektes näher betrachtet, wobei kommerzielle Implementierungen generell als noch aktiv gewertet wurden, so fern keine widersprüchlichen Informationen seitens des Herstellers vorlagen. Open-Source-, beziehungsweise Freewareprojekte wurden als aktiv bewertet, so fern ein Update der Website innerhalb der letzten drei Monate ersichtlich war.

2.2 Alleinstellungsmerkmale

Viele Implementierungen, zum Beispiel *HTTPMR* erweitern das MapReduce-Paradigma um zusätzliche Funktionalität oder neue Konzepte, wie zum Beispiel *PACT*. Es ist sicherlich von Vorteil, einige solche Implementierungen in die Fachstudie mit aufzunehmen, um mehr vergleichbare Unterschiede zu erhalten. Bei der Auswahl von Implementierungen wurde darauf acht gelegt, eine breite Streuung verschiedener Merkmalkategorien zu erreichen, beispielsweise Open Source/Proprietär oder MapReduce als Hauptfunktionalität/Zusatzfeature. Auf dieses Thema wird im Abschnitt *4 Detaillierte Betrachtung ausgewählter Implementierungen* näher eingegangen.

2.3 Lokale Testbarkeit

Im Rahmen der Fachstudie stehen nur begrenzte Ressourcen zur Verfügung, auf denen die gewählten MapReduce-Implementierungen getestet und verglichen werden können. Ein potenzieller Testkandidat muss also auf der den Bearbeitern zugänglichen Hardware lauffähig sein. Zudem muss bei kommerziellen Produkten eine kostenlose Demonstrationsversion verfügbar sein. Dieses Kriterium wurde in der Praxis eher selten angewandt. Es gab jedoch einige Fälle, in welchen die Implementierungen nur auf Proprietären Cloudplattformen lauffähig waren , oder nur mit sehr exotischen Plattformen kompatibel waren, wie zum Beispiel Elastic Phoenix auf der Linux KVM mit Nahanni shared memory system.
Im Großen und Ganzen sind die meisten Implementierungen jedoch weitestgehend plattformunabhängig. Dies liegt vor allem daran, dass sie zum größten Teil wie beispielsweise bei den Hadoop-Distributionen als Bytecode (Java, O'CAML etc.) vorliegen, oder in einer Skriptsprache wie beispielsweise Python implementiert sind.

2.4 Geschätzte Einarbeitungszeit

Da der Zeitraum zur Durchführung der Fachstudie begrenzt ist, muss sich der Einarbeitungsaufwand in die gewählten Implementierungen in Grenzen halten. Essenziell hierfür sind Existenz und Qualität der Dokumentation und eventueller Installationsanleitungen, sowie die Distributionsform.
Umfang und Qualität der Dokumentation konnten während der vorbereitenden Arbeiten natürlich nur grob abgeschätzt werden. Eine Dokumentation wurde als ausreichend umfangreich angesehen, wenn ein Implementierungsbeispiel einer MapReduce-Funktion, sowie eine Anleitung zur Jobsteuerung vorhanden waren. Darüber hinaus wurden einige Abschnitte probe gelesen um festzustellen, ob der Stil verständlich ist. Diese Kriterien leiten sich von der Dokumentation des Hadoop-Projektes ab, welches aufgrund seiner Stellung als Quasi-Standard und der generellen Qualität von Dokumentation von Projekten der Apache Foundation als Referenz genutzt wurde.
Das Fehlen einer Installationsanleitung bei Implementierungen, welche nur als Quellcode vorlagen, wurde als KO-Kriterium betrachtet, da eine Trial-and-Error-Herangehensweise durch die kombinierten Fehlerquellen Kompilierung und Installation als zu zeitaufwendig angesehen wurde.

2.5 Existenz einer Nutzerbasis

Sowohl für den praktischen Einsatz einer Implementierung, als auch für das Durchführen der Fachstudie sind solche Implementierungen besser geeignet, welche eine möglichst aktive und große Nutzerbasis besitzen. Die Definition einer relevanten Nutzerbasis erweist sich allerdings als schwierig. Während große Distributionen wie Hadoop tausende aktiver Forennutzer oder sogar Unterstützung durch einen Konzern, wie beispielsweise Nokia, besitzen, ist dies kleinen Projekten natürlich nicht möglich. Andererseits reichen wenige engagierte Nutzer aus, um die Nutzung einer Implementierung stark zu vereinfachen. Im Rahmen der Vorauswahl wurde eine relevante Nutzerbasis als solche definiert, welche eine Mitgliederzahl von mindestens einhundert registrierten Nutzern aufwies, und deren letzter Forenbeitrag nicht länger als 3 Wochen zurück lag.

2.6 Medienpräsenz/Popularität

Wie bereits im Punkt *2.1 Entwicklungsaktivität* angesprochen, soll die Fachstudie nach Möglichkeit für das Publikum relevante MapReduce-Implementierungen behandeln. Da die tatsächliche Verbreitung einer gegebenen Implementierung nur schwer analysiert werden kann, kann diese nur abgeschätzt werden. Hierzu bietet sich das Such-/Nachrichtenvolumen von Google Trends an (siehe *Abbildung 2.6.1*).

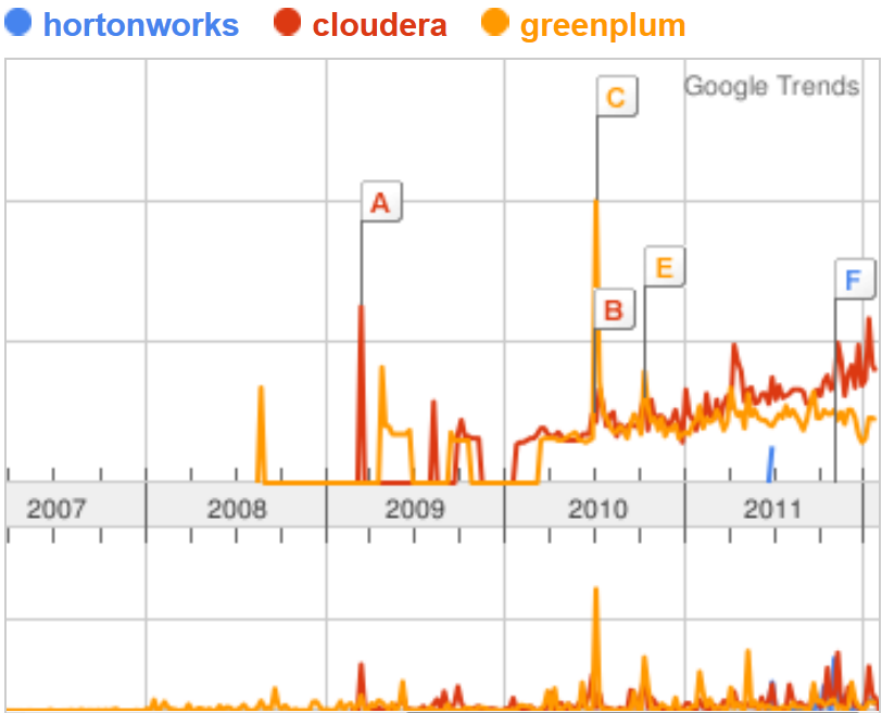

Abbildung 2.6.1: Google Trends Ergebnisse für hortonworks, cloudera, greenplum

Dieses Kriterium ist jedoch nur für die besonders populären Implementierungen (hauptsächlich die diversen Hadoop-Distributionen) von Nutzen, da Forschungsprojekte außerhalb ihres Feldes nur wenig Medienaufmerksamkeit erhalten. Deshalb wurden bei der Auswahl der Implementierungen auch einfache Suchen in Suchmaschinen herangezogen, da im Internet auftauchende Fragen zu Installation und Gebrauch einer Implementierung auf eine reale Nutzerbasis schließen lassen. Darüber hinaus beinhalten einige Projektwebsites eine Liste von Institutionen und Firmen, welche laut dem Entwickler die Implementierung verwenden. Auch dies floss in die Schätzung der Popularität mit ein.

2.7 Ergebnis der Vorauswahl

Nach der Berücksichtigung der Auswahlkriterien schrumpfte der Menge der Implementierungen auf folgende 14 Kandidaten:

Hadoop-basierend
Apache Hadoop, MapR, Cloudera und Cascading

Nicht Hadoop-basierende Implementierungen von MapReduce
Basho Riak, Aster Data SQL-MapReduce, MongoDB, Nokia Disco, Spark und Myspace Qizmt

Implementierungen mit MapReduce ähnlichen Programmiermodellen
Gearman, Stratosphere, GridGain und Sector/Sphere

Dabei wurde, wie beispielhaft in *Abbildung 2.7.1* zu sehen, mithilfe von zuvor definierten Farben jede einzelne Implementierung, sowie der jeweilige Bereich markiert. Dabei wurden Bereiche, welche durch die Implementierung vom Team als nicht ausreichend bewertet wurden, mit der Farbe O*range* markiert. Diese KO-Kriterien sorgten dann für eine Streichung der Implementierung aus der Liste der möglichen Kandidaten und wurde zur Folge *gelb* markiert. Die so nicht aussortierten Implementierungen wurden G*rün* markiert. Die komplette KO-Tabelle befindet sich in *Anhang 1.3.*

MapReduce Implementations	Popularity	Documentation / Coverage / Community	Installation / Difficulty
Hadoop-based distributions			
Apache Hadoop	very high	available / very high / very high	Installation guide /
HortonWorks	low	available / very low / very low	none
MapR	high	available / high / high	Installation guide /
Platform MapReduce	very low	available / mediocre / very low	none
Cloudera	high	available / high / very high	Installation guide /
Greenplum MapReduce	very low	available / very low / mediocre	Installation guide /
Apache Hive	mediocre	available / high / mediocre	Installation guide /
Apache Pig	mediocre	available / mediocre / mediocre	Installation guide /
Dumbo	very low	available / low / low	Installation guide /
Rhipe	very low	available / very low / very low	Installation instructions /
Cascading	low	available / high / high	Installation guide /

Abbildung 2.7.1: KO-Tabelle für Hadoop-basierende Implementierungen

3 Bewertung der MapReduce-Implementierungen

In diesem Abschnitt werden die Implementierungen der Vorauswahl auf die unten beschriebenen Kriterien untersucht und bewertet. Dabei wurde folgendes Bewertungsschema für die Bewertung erarbeitet.

3.1 Bewertungsschema für den ersten Vergleichstest

Die Kategorien lauten: **MapReduce-Implementierung, Datenbank und Dateisystem, Dokumentation, Installation, Besonderheiten**
Bis auf die Besonderheiten wird jede Kategorie in Basis- und Bonuspunkte unterteilt. Dabei können Basispunkte durch folgende Symbole bewertet werden:

0 Vorhanden

\+ Positiv auffällig

\- Mangelhaft oder nicht vorhanden

Boni und Besonderheiten sind aus sich heraus hervorzuhebende Eigenschaften und Alleinstellungsmerkmale und erhalten daher jeweils ein Plus.

MapReduce - Implementierung (maximal 10 Punkte):

Basis (maximal 4 Punkte):

- Vorhandensein des nativen "klassischen" MapReduce
- Flexibilität (z.B. mehr als eine Job-Implementierungssprache wird unterstützt, Jobs können während der Laufzeit angepasst werden, Master-Knoten können die Last während der Laufzeit neu verteilen)
- Skalierbarkeit (z.B. erhöhen zusätzliche Rechner im Cluster die Performanz der MapReduce Jobs bzw. senken die Arbeitslast der Worker)
- Einfachheit (z.B. verwendete Skriptsprache, Konfiguration, wie einfach können MapReduce-Aufträge erstellt werden)

Boni: (maximal 6 Punkte)

- Analyse der MapReduce-Aufträge
- Vorhandensein von GUIs für wichtige Programmteile
- Verwendung von MapReduce ähnlicher Implementierungen
- Sicherheit (z.B. Jobs bleiben bei Ausfall erhalten, kein Single-Point-of-Failure)
- Gegebenenfalls weitere Nennungen

Datenbank/Dateisystem (maximal 7 Punkte):

Basis (maximal 4 Punkte):

- Vorhandensein (integriertes verteiltes Dateisystem/Datenbank)
- Skalierbarkeit (z.B. können Rechner während des Betriebs dem Cluster hinzugefügt werden)
- Sicherheit (z.B. Fehlertoleranz bei Hardware-/Datenausfall)
- Einfachheit (z.B. Konfiguration und Einrichtung eines funktionierenden Clusters / einer Cloud)

Boni (maximal 3 Punkte):

- Flexibilität (z.B. mehr als ein Dateisystem/Datenbank wird unterstützt)
- Management (z.B. Überwachung und Bedienung/Steuerung mittels GUI)
- Schnittstellen (z.B. fremde Tools/Programme werden unterstützt)

Dokumentation (maximal 5 Punkte):

Basis (maximal 3 Punkte):

- Vorhandensein der Bedienungsanleitung (z.B. Online, PDF, etc.)
- Ausführlichkeit (z.B. Beispiele, Bilder)
- Komfort (z.B. Bedienung, Querverweise, Suche, Stichwortverzeichnis)

Boni (maximal 2 Punkte):

- weitere Ressourcen (z.B. Videos, Tutorials)
- aktive Nutzerbasis (z.B. Forum, Communiy-Sites, Mailing-List)
- direkter Entwicklersupport

Installation (maximal 5 Punkte):

Basis (maximal 4 Punkte):

- Vorhandensein einer Installations- und Konfigurationsanleitung
- unterstütztes und getestetes OS (z.B. Linux, Windows, Mac)

Boni (maximal 1 Punkt):

- Einfachheit (z.B. fertige Binary, einfaches Entpacken einer ZIP-Datei oder Installer)

Besonderheiten (maximal 5 Punkte):

Alle erwähnenswerten und sinnvollen Features, die sich grundlegend von anderen Implementierungen abheben, erhalten in dieser Kategorie ein zusätzliches Plus. Negativ auffallende Besonderheiten führen nicht zu einem Minus.

3.2 Hadoop-basierende MapReduce-Implementierungen

3.2.1 Apache Hadoop

Apache Hadoop ist ein unter der freien Apache-Lizenz veröffentlichtes Software-Framework *[HAD]*, welches auf den ursprünglichen Veröffentlichungen [DNS06] zu MapReduce und Google File System basiert. Obwohl es anfänglich für die Infrastruktur der freien Suchmaschine Nutch entwickelt wurde, ist heute Yahoo! einer der größten Beitragsleister des Projektes. Yahoo! verwendet Hadoop für den Großteil seiner Firmeninfrastruktur. Hadoop nutzt Google's MapReduce-Ansatz, welcher Cluster mit mehreren Tausend Knoten ansprechen und nutzen kann. Jedes Problem, das aufgeteilt und unabhängig voneinander berechnet werden kann, bietet sich als eine MapReduce-Funktion an. Da Hadoop in Java geschrieben wurde, können auch entsprechende Programme aufgerufen und ausgeführt werden. Durch Erweiterungen wie Apaches Hive und Pig entsteht z.B. die Möglichkeit, Daten in einer SQL-ähnlichen Sprache abzufragen, mittels komplexer Berechnungen zu analysieren und zu verändern.
Um Daten mittels MapReduce bearbeiten zu können, müssen diese in Hadoops eigenem Dateisystem (HDFS) gespeichert vorliegen. Dieses kann besonders große Datenmengen auf großen Clustern verteilt speichern und bietet unter anderem Fehlertoleranz und Skalierbarkeit und kann leicht erweitert werden. Mit HBase bietet Hadoop dazu eine Datenbank, die auf HDFS aufbaut.
Der Bekanntheitsgrad von Hadoop basiert neben der Verwendung von robusten Google-Technologien vor allem auf einer ausführliche Dokumentation, welche die Einrichtung und die Handhabung des Frameworks deutlich vereinfacht. Auf der Homepage von Hadoop befinden sich die Installationsdateien und eine Anleitung für die Einrichtung auf einem als auch auf mehreren Rechnern. Für den produktiven Einsatz wird dabei nur Linux unterstützt, für Entwicklungs- und Testzwecke mit Einschränkungen auch Windows.
Die Architektur von Hadoop-Clustern, so wie dessen Ausführung von MapReduce-Jobs, ist eine exakte Umsetzung des ursprünglichen MapReduce-Paradigmas, wie es in [DNS06] beschrieben wurde. Dadurch eignet sich Hadoop für die genauere Betrachtung, weckt jedoch auch kein besonderes Interesse durch eventuelle kreative Ansätze. Die Skalierungsmöglichkeiten sind ebenfalls im Bereich des von einer MapReduce-Implementierung Erwarteten.
Das Hadoop zugrunde liegende Dateisystem, HDFS, entspricht ebenfalls den Erwartungen an ein modernes verteiltes Dateisystem: Der Ausfall einzelner Knoten kann durch Datenreplikation kompensiert werden, das Dateisystem lässt sich zusammen mit dem MapReduce-Cluster skalieren und es existiert mit der *Name Node*, dem Speicher für Metadaten, nur ein einzelner Single Point of Failure. Einzig die Einrichtung und Konfiguration, welche ausschließlich über Konsolenbefehle und Textdateien möglich ist, fiel negativ auf.
Die Dokumentation des Hadoop-Projektes ist online als Hypertext verfügbar. Sie ist durch Inhaltsverzeichnis und Querverweise ordentlich strukturiert, ihr Umfang selbst für ein Open Source Projekt überdurchschnittlich. Für eine genauere Betrachtung spricht überdies die Tatsache, dass die hohe Verbreitung von Hadoop zu einer äußerst aktiven und hilfreichen Nutzerbasis geführt hat, welche ihre Expertise in zahlreichen Foren, Wikis und Blogs zugänglich macht. Die Installations- und Konfigurationsanleitungen der offiziellen Projektseite sind akzeptabel. Obwohl sie sich auf eine veraltete Version von Hadoop beziehen, sind sie immer noch brauchbar.
Apache Hadoop ist nur unter Linux offiziell lauffähig. In der Theorie ist es möglich, Knoten mittels Cygwin auf Windowsplattformen zu betreiben, hiervon wird jedoch seitens des Hadoop-Projektes außer zu Testzwecken abgeraten. Auf der Website sind neben dem Quellcode auch fertige

Binärdistributionen erhältlich. Dies ist erwähnenswert, da dies gerade im Bereich der Open Source Projekte keine Selbstverständlichkeit darstellt.

Besonderheiten
Viele MapReduce-Implementierungen basieren auf Hadoop und erweitern dieses um die Bedienung weiter zu vereinfachen und die Nutzungsmöglichkeiten zu erweitern. Zu Erweiterungen unter der Verwaltung der Apache Foundation zählen unter anderem *Hive*, *Pig* und *HBase*. *Apache Hive* erweitert Hadoop um eine Abfragesprache namens HiveQL, welche auf SQL basiert und eine ähnliche Syntax besitzt. Intern werden Abfragen von einem Compiler in einen gerichteten azyklischen MapReduce-Grafen übersetzt, welcher Hadoop zur Ausführung übermittelt wird. Daten müssen hierfür in einem Dateisystem wie HDFS oder HBase vorliegen und können durch eigene QL-Erweiterungen extrahiert, transformiert, geladen und analysiert werden. *Apache Pig* ist eine Plattform die, ähnlich wie Hive, eine Erweiterung von Hadoop darstellt. Mit einer neuen Sprache namens *Pig Latin* können prozedurale Programme geschrieben und von einem Compiler zu MapReduce-Anweisungen übersetzt werden, welche Hadoop verstehen und auf HDFS ausführen kann. Standardmäßig stellt *Pig* Operationen zum Analysieren und Verändern von Daten bereit, ermöglicht dem Benutzer aber auch sogenannte User Defined Functions (UDF) zu programmieren und mit einfachen Pigskripten komplexere Berechnungen durchzuführen. Distributionen wie die von uns untersuchten *MapR* und *Cloudera* bieten zusätzlich grafische Oberflächen, Management- und Überwachungsfunktionen sowie erweiterte Schnittstellen für Hadoop Cluster.

3.2.2 MapR

MapR ist eine proprietäre Hadoop Distribution der gleichnamigen Firma im kalifornischen San Jose*[MAR]*. Schwerpunkte der Distribution sind laut Hersteller eine Steigerung der Performanz, die Vermeidung des Single-Point-of-Failures und eine erleichterte Handhabung für Nutzer. Ein Teil des MapR-Codes fließt in das ursprüngliche Apache Projekt zurück, so hat MapR seit seiner Gründung im Jahre 2009 Beiträge zu den Teilprojekten *HBase*, *Hive* und *ZooKeeper* geleistet. Nichtsdestoweniger finden sich viele Features, wie zum Beispiel NFS-Anbindung und Snapshots, nicht im Hauptprojekt wieder. *MapR* wird parallel zu *Hadoop* und dessen Programmerweiterungen installiert und benützt daher auch *Hadoop's* MapReduce-Implementierung mit all ihren Möglichkeiten der Erweiterbarkeit durch z.B. *Apache Hive* oder *Pig*. MapR existiert einmal als freie Version M3 mit eingeschränkter Funktionalität und einmal als kostenpflichtige Version M5, die 30 Tage getestet werden kann.
Die Architektur der MapReduce-Implementierung von *MapR* entspricht jener von *Hadoop*, wurde allerdings um einige für die weitere Betrachtung interessante Aspekte erweitert. Zum Beispiel müssen Jobs nicht mehr strikt sequentiell ablaufen, d.h. spontane kleine Aufträge können auf Wunsch kurzzeitigen Vorrang im Job-Queue erhalten. Dies erhöht in entscheidendem Maße die Flexibilität der Implementierung. Ein weiterer Pluspunkt unter dem Aspekt der Flexibilität ist die dezentrale Auftragsüberwachung. Falls diese abstürzt, wird sie dynamisch auf einem anderen Rechner neu gestartet und alle Aufträge neu verknüpft. In einem klassischen Hadoop-System würden alle Aufträge verloren gehen, dies wird in *MapR* vermieden. Dies ist unter dem Aspekt der Sicherheit interessant. Das starten von Jobs ist unter *MapR* ähnlich simpel gehalten wie unter Hadoop, es ist keine Vorkonfiguration notwendig. MapR besitzt zur Auftragssteuerung ein eigenes System namens *MapR Control System*. Dieses stellt einen interessanten Bewertungsaspekt dar, vor allem in Bezug darauf, inwiefern sich dies als Vorteil gegenüber der Steuerung eines reinen Hadoop-Clusters erweist.
Das Dateisystem von *MapR* entspricht im Wesentlichen *HDFS*. Durch Dateisystemoptimierungen

wurde jedoch die Geschwindigkeit der Abarbeitung von MapReduce-Aufträgen um den Faktor zwei bis fünf erhöht [MAS]. Eine interessante Besonderheit des *MapR*-Dateisystems ist eine Technik namens *Direct Access NFS.* Über diese können Hadoopcluster direkt als Netzwerkfestplatte eingebunden und über den Dateibrowser des Betriebssystems bearbeitet werden. Ein einmal beschriebener Hadoop-Cluster kann nicht mehr erneut beschrieben werden, sondern nur noch Lesezugriffe werden erlaubt. Mit NFS dagegen können Dateien innerhalb des Clusters einzeln als auch gleichzeitig konkurrierend geschrieben und gelesen werden. Betriebssystemspezifische Funktionen wie Auflisten, Sortieren, Drag-and-Drop von Daten in das Cluster hinein und hinaus sind zusätzlich möglich. Interessant für eine weitergehende Betrachtung sind auch die so genannten *MapR Volumes.* Mit Hilfe dieser können Cluster partitioniert und in eine Baumstruktur eingebunden werden um Daten zu organisieren und zu sichern, zum Beispiel durch festgelegte automatische Spiegelungen und Snapshots. Ebenso können Datenrechte und Quotas verteilt und die Häufigkeit der Benutzung verfolgt werden. Die Anzahl von Dateien in einem Cluster sind im Gegensatz zu Hadoop unbegrenzt, da nicht ein Rechner alle Dateien kennen muss. Damit existiert auch kein Single Point of Failure. Diese Eigenschaft der Dateisystemarchitektur ist unter den Aspekten der Sicherheit und der Flexibilität interessant.
Sowohl die Installation, als auch die Nutzung und Fehlerbehandlung von *MapR* ist auf der Homepage sehr detailliert dokumentiert. Beispielvideos erklären zusätzlich einige Grundkonzepte der Handhabung und erleichtern den Einstieg. Die Dokumentation ist auf die einzelnen Module aufgeteilt. Obwohl dies dazu führt, dass sämtliche Module ausführlich dokumentiert sind, erweist sich die Handhabung als etwas umständlich. Ähnlich *Hadoop* besitzt *MapR* eine sehr aktive Nutzerbasis, auf dessen Expertise in vielfältiger Form zurückgegriffen werden kann. Fertige Binärdaten so wie der Quelltext können von der Homepage zur Installation heruntergeladen werden und erfordern ein 64-Bit Betriebssystem basierend entweder auf Ubuntu, CentOS oder Red Hat EL. Das Java JDK 6 wird zum Ausführen benötigt.

Besonderheiten
Beide MapR Versionen erlauben die Einsicht und Bedienung von Hadoop-Clustern über eine grafische Oberfläche. Diese nennt sich MapR Control System (MCS) und gibt visuell Auskunft über Aktivität und Ressourcen im Cluster sowie Bericht über Gesundheitsstatus und Auslastung einzelner Rechner. Durch Filter können zum Beispiel einzelne Komponenten aufgespürt und zu definierten Gruppen zugewiesen werden. Erhöhte Dateisicherheit und 24-Stunden-Support wird mit der kommerziellen Version angeboten.

3.2.3 Cloudera

Cloudera ist sehr ähnlich wie *MapR* eine Closed Source Distribution für eine zweistellige Anzahl von Open-Source-Programmen*[CLD]*. Darunter verwendet es als Basis *Apaches Hadoop* und ergänzt es mit Entwickler-, Management- und Administrationstools, die unter anderem auch *Apache Hive*, *Pig* und *HBase* sowie etliche weitere Tools enthalten. Es steht eine kostenfreie Version sowie eine kostenpflichtige Version zu zum Download bereit. Die kostenlose Version arbeitet auf maximal 50 Rechnern zusammen, und ihr fehlen einige Datenanalyse-, Überwachungs- und Auswertungs-Tools. Um dem professionellen Anspruch gerecht zu werden, bietet *Cloudera* Firmen Trainingsstunden und Support an.
Cloudera übernimmt im Kern Apache Hadoop in unveränderter Form. In Folge dessen findet sich kein Grund für eine weitergehende Betrachtung auf Grund der Architektur. *Cloudera Desktop* ist die Bezeichnung für die grafische Oberfläche, über die sowohl der Zugriff auf alle genutzten Programme, als auch auf das Dateisystem ermöglicht wird. Dies erhöht in entscheidendem Maße

den Bedienkomfort. *Cloudera* bietet ein Checkpointing-System, welches die Flexibilität in Bezug auf Wartung zur Laufzeit so wie die Robustheit des Systems erhöht. Zudem werden Zahlreiche externe Schnittstellen geboten, mit Hilfe derer die Funktionalität erweitert werden kann. Da *Apache Hadoop* ein unveränderter Bestandteil der Cloudera Distribution ist, wird ebenfalls auf dessen Dateisystem *HDFS* zurückgegriffen, welches als primäres Speichersystem dient und jede Komponente mit Daten beliefert. *Cloudera* erweitert das Dateisystem mit dem *HA Name Node*-Konzept, dessen Ziel es ist, die Verfügbarkeit von verteilten Systemen mithilfe eines Aktiv/Passiv-Systems zu erhöhen. Dabei wird ein *Hot-Standby*-Knoten eingerichtet, der bei Bedarf für einen ausfallenden Knoten ohne Zeit- oder Ressourcenverluste einspringen kann. Dies entfernt den Single Point of Failure eines reinen *Hadoop*-Clusters. *Cloudera* bietet auch Monitoringwerkzeuge zum Überwachen des *HDFS*-Clusters.
In einer ausführlichen Ressourcendatenbank werden professionelle Paper, Videos, Artikel, Case Studies und FAQs angeboten. Dies ist für eine kommerzielle Distribution überdurchschnittlich. Die Online-Dokumentation liegt nicht als geordneter Hypertext vor, dies erschwert die Suche nach einem bestimmten Thema stark. Sie ist jedoch umfassend und wird unter anderem zum Download als PDF-Dateien zu Verfügung gestellt. Durch die Installation hilft eine ausführliche und umfangreiche PDF, die auch Auskunft über Konfiguration und Überwachung der Cloud liefert.
Unterstützt werden seit der neuesten Generation zahlreiche Unix Betriebssysteme wie zum Beispiel CentOS, Debian oder Ubuntu. Auf Windows und Max OS läuft die Software jedoch nicht. *Cloudera* ist ausschließlich als Binärdistribution erhältlich, was jedoch den Erwartungen an eine proprietäre Distribution entspricht.

Besonderheiten
Cloudera integriert eine Vielzahl von Tools welche die Software interessant machen. Neben Erweiterungen von Datenbank- und Dateisystem werden auch GUI-Komponenten integriert, zum Beispiel für die Interaktion mit Apache's Hadoop. Zahlreiche Module sollen eine eine komfortablere und flexiblere Arbeit mit MapReduce-Aufträgen ermöglichen.

3.2.4 Cascading

Cascading ist eine Open Source Java-Abstraktionsebene für Apache Hadoop *[CAS]*. Sein Nutzen besteht in der Möglichkeit, komplexe Workflows auf einem *Hadoop* Cluster auszuführen, ohne sich mit den MapReduce-Paradigma zugrunde liegenden komplexen Strukturen auseinanderzusetzen. Hierzu kann Funktionalität in jeder beliebigen *JVM*-kompatiblen Programmiersprache implementiert werden. Zusätzlich hierzu wurden für den Einsatz mit *Cascading* die domänenspezifischen Sprachen *Cascading.jruby* und *Cascalog* entwickelt. Die Nachfolgeversion 2.0 der aktuellen Version (1.2) befindet sich gerade in Entwicklung. Da es sich nur um eine Bibliothek handelt, die direkt auf einer *Hadoop*-Installation läuft, wird auf alle Funktionen Hadoops zurückgegriffen. Hierzu zählt auch MapReduce, jedoch wird mit Hilfe der Möglichkeit Befehle zu kapseln die Erstellung und Handhabung der MapReduce-Aufträge vereinfacht. Abgesehen davon unterscheidet sich *Cascading* nicht wesentlich von Hadoop, und ist somit nicht für eine weitergehende Betrachtung interessant. Da *Cascading* auf *Hadoop* aufsetzt, wird als Dateisystem ein unverändertes *HDFS* eingesetzt.
Es steht eine akzeptable Benutzerdokumentation online zur Verfügung. Sie wurde allerdings das letzte mal im Jahre 2010 aktualisiert (Stand 2.5.2012). Zudem gibt es viele Beispiele und Querverweise. Auf der Homepage von *Cascading* befinden sich die Installationsdateien als Tarball. Die Installations- und Konfigurationshilfe beschränkt sich auf eine simple Textdatei.

Besonderheiten
Durch Kapselung von komplexeren Befehlsreihen können Aufgaben mithilfe einfacher JAR-Dateien ausgeführt werden. Durch seine einfache Handhabung bietet sich Cascading vor allem für schnelle und einfache Aufgaben an, auch weil die Installation und Konfiguration keinen größeren Aufwand erforderlich macht.

3.3 Nicht auf Hadoop basierende Implementierungen von MapReduce

3.3.1 Basho Riak

Riak ist ein erstmals 2009 unter der Apache 2.0 Lizenz veröffentlichtes NoSQL-Datenbanksystem, welches auf dem Dynamo-Prinzip aus dem Hause Amazon basiert*[RIA]*. Es bietet integrierte MapReduce-Funktionalität mit nativer Unterstützung von JavaScript und Erlang, zusätzliche Sprachen, wie zum Beispiel Python, Ruby oder PHP, können jedoch über zusätzliche Treiber genutzt werden. Riak ist in der Wirtschaft weit verbreitet, unter anderem zählen Comcast, AOL und Mozilla zu seinen Nutzern. Für Geschäftskunden ist eine kommerzielle Version mit vollständigem Funktionsumfang und zusätzlichen Support auf persönliche Anfrage verfügbar. Das System besitzt getrennte MapReduce-Verfahren zur Abfrage und Verarbeitung von Daten. Die aktuelle Implementierung der beiden, welche unter dem Überbegriff *Riak Pipe* gekapselt, basiert auf der Programmiesprache Erlang, und bietet laut Hersteller neben erhöhter Stabilität vor allem mehr Geschwindigkeit. Bezüglich der nativen Ausführung von MapReduce-Aufträge ist Riak erwähnenswert flexibel: Abfragen können in Erlang und Javascript geschrieben werden, während Funktionen, die auf Ersterem basieren, vollständigen Zugang zur Riak-Erlang-Schnittstelle haben. Javascript-Funktionen werden mit Hilfe von Mozillas Spidermonkey-Engine übersetzt und sind abhängig von ihrer Komplexität kaum langsamer in der Ausführung als ein in Erlang geschriebener Auftrag*[RIA]*. Zur Verbindung von Riak mit einer präferierten Programmiersprache stehen zusätzlich sechs verschiedene Client-Bibliotheken für die Sprachen Erlang, Java, JavaScript, PHP, Python und Ruby zur Verfügung, was die Flexibilität der Implementierung weiter erhöht. Die Steuerung der Aufträge ist erfreulich einfach: Es steht eine HTTP-API zur Verfügung, mit der der Cluster gesteuert werden kann.
Riak besitzt ein eigenes Dateisystem. Positiv fällt auf, dass es in *Riak*-Clustern keine zentrale Steuereinheit und somit keinen Single Point of Failure gibt. Jeder Rechner in einem Cluster wird als Knoten bezeichnet und besitzt weitere virtuelle Knoten, die sogenannte *Buckets* mit einem Wert speichern können. Alle *Buckets/Keys* erhalten einen einzigartigen 160bit-Binärwert, die zusammengefasst in einer Protokolldatei als eine Art Informationsring vorliegen. Dieser wird in Echtzeit entsprechend der Anzahl an Knoten partitioniert und diesen zugewiesen. Als besonders flexibel erweist sich *Riak* dadurch, dass die Anzahl an Replikationen, die Vereinigungsstrategie und das Fehlermodell während des Betriebs gewählt und verändert werden können. Auf gleiche Weise können weitere Rechner angeschlossen werden, um entweder die Datensicherheit zu erhöhen oder mehr Speicherkapazität und Gesamtperformanz zur Verfügung zu stellen. Positiv fällt zu dem die grafische Benutzeroberfläche *Riak Control* auf, mit der Cluster und Knoten überwacht und gesteuert werden können. Informationen über den Gesundheitsstatus, die Auslastung und Verfügbarkeit der Knoten sowie die ausgeführten MapReduce-Aufträge können eingesehen werden. Überdies sind die HTTP-Schnittstellen interessant, über welche *Riak* mit weiterer Software verbunden werden kann.
Die Installation und Nutzung von *Riak* ist sowohl auf der Homepage, als auch im Quellcode

detailliert dokumentiert. Ein Video erklärt zusätzlich die Bedienung von *Riak Control*. Inoffizielle Nutzervideos, welche die Funktionalität von Riak erklären, sind ebenfalls erhältlich. Zusätzlich dazu stehen Tutorials zur Verwendung der Client-Bibliotheken zur Verfügung. Eine weitere Informationsquelle ist die breite Nutzerbasis, welche ihr Wissen um *Riak* auf Mailing Lists, Blogs und diverse Social-Media-Plattformen zugänglich macht.
Riak kann von der Homepage als freie Open-Source-Version mit eingeschränkter Funktionalität heruntergeladen werden. Sowohl Quelltext als auch eine Binärdistribution sind verfügbar. Unterstütze Betriebssysteme sind Linux und MacOS X in 32 und 64 Bit.

Besonderheiten
Besondere Features wie eine HTTP-Schnittstelle, eine integrierte Volltextsuche, eine zweite Indizierungsebene, sowie Unterstützung durch die Riak Developement Community bilden die Open-Source-Basis. Die kostenpflichtige Enterprise-Version erweitert den Funktionsumfang um Sicherungskopien, SNMP-Überwachungsagenten und eine direkte Riak-Entwicklerbetreuung, die 24 Stunden zur Verfügung steht. Beliebige Programmiersprachen können durch vorhandene oder selbst erstellte Clients genutzt werden.

3.3.2 Aster Data SQL-MapReduce

Aster Data SQL-MapReduce ist eine proprietäre Implementierung des MapReduce-Paradigmas auf *nCluster*, Aster Data's hauseigener SQL-Cluster Architektur*[AST]*. Sie bietet eingebettete MapReduce-Funktionalität in SQL-Queries zur Manipulation von Datenbankinhalten.
SQL-MapReduce nutzt eine interessante Interpretation des MapReduce-Paradigmas: Es teilt das MapReduce-Schema in zwei unabhängige Teile auf. So genannte *Row Functions* bearbeiten jeweils eine einzelne Tabellenzeile, entsprechen also dem Map-Teil eines traditionellen MapReduce-Aufträge. Die von Aster als *Partition Functions* bezeichneten Funktionen werden auf frei definierbare Gruppen von Zeilen angewendet, entsprechen also dem Reduce anderer Implementierungen. Die Ausgabe einzelner *Row/Partition-Functions* werden von der zugrunde liegenden *nCluster*-Engine als SQL-Tabellen behandelt, lassen sich also als Arbeitsgrundlage für weitere *Row-/Partition-Functions* sowie für konventionelle SQL-Queries verwenden. Somit werden flexible Generalisierungen des MapReduce-Paradigmas möglich, beispielsweise Map-Map-Map-Reduce oder Map-Reduce-Reduce. Die *Row/Partition-Functions* lassen sich in einer Vielzahl von Programmiersprachen entwickeln, zum Beispiel mit Java, Python, C# oder Ruby. Aster Data bietet hierzu Funktionsprototypen/Interfaces, sowie eine Installationsroutine an, mithilfe jener die Funktionen in *nCluster* integriert werden können.
SQL-MapReduce basiert auf *nCluster,* einem frei skalierbaren, laut Hersteller bis zu mehreren hundert Terrabyte großen[AST], Datenbankcluster der Firma Aster Data. *nCluster* verteilt eine SQL-Datenbank auf mehrere Knoten. Die Architektur bietet nach außen eine standard-SQL-Schnittstelle, welche gängige SQL-Interfaces (JDBC, ODBC) unterstützt. Die Koordination der einzelnen Knoten geschieht, analog zum klassischen MapReduce, über einen so genannten *Queen Node*. Diese nimmt SQL-Queries so wie *Row/Partition-Functions* entgegen, verteilt sie auf die Workerknoten, und fügt deren Ausgabe zusammen. Allerdings übernimmt diese Konfiguration auch den klassischen *Single Point of Failure* des MapReduce-Paradigmas und dessen Auswirkungen auf die Datensicherheit. Im Allgemeinen lässt sich über die Fehlertoleranz keine Aussage treffen, da dieser Teil der Architektur nicht öffentlich dokumentiert ist.
Dokumentation sowie Installation sind zu diesem Zeitpunkt nicht überprüfbar, da diese nur zusammen mit einem Kaufvertrag oder einer autorisierten Demonstrationsversion ausgeliefert werden. Somit können diese Punkte nicht weiter betrachtet werden.

Besonderheiten
Die MapReduce-Implementierung kann direkt auf Datenbanken operieren und ermöglicht die Integration von SQL-Query-Jobs.

3.3.3 MongoDB

MongoDB ist ein erstmals 2009 unter der AGPL-Lizenz veröffentlichtes NoSQL-Datenbanksystem der Firma 10gen *[MDB]*. Ursprünglich von 10gen als Teil einer *Platform-as-a-Service*-Lösung entwickelt, ist es heute als eigenständiges Projekt erhältlich. Im Gegensatz zu klassischen Datenbanken speichert MongoDB Daten nicht in Tabellen, sondern als JSON-Dateien mit dynamischen Schemata (BSON). Zu den prominenten Nutzern von MongoDB zählen unter anderem MTV und Craigslist.
MapReduce-Aufträge werden als Datenbankkommandos implementiert, können also nicht direkt in Queries integriert werden. Sie verarbeiten einzelne Collections: Gruppen von BSON-Dateien, das MongoDB-Pendant zu Tabellen. Dabei werden die Aufträge in MongoDB als JavaScript geschrieben, welcher auf dem MongoDB-Server ausgeführt wird. Dies schränkt die Flexibilität der MapReduce-Funktionalität ein, da andere Sprachen nicht zur Implementierung von Aufträge verwendet werden können. Parallelität wird hier über horizontale Fragmentierung (*Sharding*) erreicht. Allerdings bezieht sich dieser nur auf einzelne Shards, nicht aber auf einzelne BSON-Dateien, sofern der Chunk auf dem Shard-Server aus mehr als nur einem BSON-Dokument besteht. MongoDB besitzt eine große Anzahl an Monitoringwerkzeugen, sowie Plugins für Zahlreiche Programme von Drittanbietern, welche diese Implementierung für die weitere Betrachtung interessant machen.
MongoDB nutz das native Dateisystem seines Hostbetriebssystems. Da die maximale Größe von BSON-Dokumenten jedoch begrenzt ist (4MB bis Version 1.6, 16MB ab Version 1.7/1.8), ist MongoDB in der Lage, große Dateien auf mehrere BSON-Dateien aufzuteilen. Hierzu wird eine öffentlich zugängliche Spezifikation namens GridFS genutzt, welche das Aufteilen von großen Dateien in Collections aus BSON-Dokumenten regelt. GridFS ist jedoch kein Dateisystem im eigentlichen Sinne, da nur Organisation und Struktur des Dateiinhalts spezifiziert werden, jedoch nicht die eigentliche Ablage auf dem Datenträger. Aus diesem Grund ist auch die Einrichtung eines MongoDB-Clusters besonders einfach, da der Cluster nicht erst formatiert werden muss.
Die Dokumentation ist im Allgemeinen sehr gut geschrieben. Auf der Website finden sich zahlreiche gut geschriebene Tutorials und Beispiele für das komplette Anwendungsspektrum der Software, so wie spezialisierte technische Hilfestellungen für Administratoren und Entwickler. Die Nutzerbasis des Projektes ist sehr aktiv, es stehen mehrere Foren und inoffizielle Tutorials zur Verfügung.
MongoDB ist als Binärdistribution für OSX, Microsoft Windows und viele gängige Linux Distributionen erhältlich. Zusätzlich dazu existiert eine Building-Page mit Hilfestellung zum Kompilieren für andere unixoide Betriebssysteme. Detaillierte Installationsanleitungen sind ebenfalls vorhanden.

Besonderheiten
Interessant im Bezug auf MapReduce-Aufträge ist das von MongoDB angebotene *incremental MapReduce*. Im Falle von sich schnell ändernden Collections erlaubt dieses bei wiederholtem Ausführen der Aufträge nur neue Einträge zu verarbeiten und die neuen Ausgabedaten in die bereits bestehenden zu integrieren. Die Unterscheidung zwischen neuen und bereits verarbeiteten Daten geschieht hier über einen in den Job eingebetteten Query.

3.3.4 Nokia Disco

Nokia Disco ist ein im Jahr 2008 von Nokia gestartetes Open-Source-Projekt *[NOD]*. Die Entwicklung begann, weil Nokia nach anfänglichen Versuchen einer Hadoop-Modifikation zu dem Schluss kam, dass dieses seinen Anforderungen nicht genügen würde. Als Implementierungssprache wurde Erlang gewählt. Nokia bezeichnet die MapReduce-Implementierung von Disco als Master-Slave-Architektur. Diese erweitert das Master-Worker-Prinzip der klassischen MapReduce-Implementierungen um so genannte *Slaves*, welche die Worker auf einem Knoten verwalten, sowie ihnen Aufträge zuteilen. Der Cluster kann sich somit auch selbst skalieren, in dem neue Worker auf einem Slave gestartet werden. Die Standardanzahl von Workern auf einem einzelnen Knoten ist frei konfigurierbar. Somit erfüllt *Nokia Disco* die Erwartungen an eine MapReduce-Implementierung. Die Mögliche Selbstskalierung macht sie sogar etwas flexibler als die Standardarchitektur. Innerhalb eines Clusters sind mehrere *Master* möglich, diese besitzen jedoch keine gemeinsame Job-Queue. Neue Knoten können zur Laufzeit über ein Webinterface hinzugefügt werden. Disco versucht, so weit als möglich, Datenlokalität beizubehalten. Für den Fall, dass dies nicht möglich sein sollte, wird vom *Master* auf jedem Knoten ein HTTP-Server gestartet, so dass benötigte Daten über dieses Protokoll zwischen den Knoten ausgetauscht werden können. Ein gewisses Maß an Flexibilität bietet die Tatsache, dass Arbeitsaufträge sowohl in Python, als auch in Objective CAML implementiert werden können.
Disco benutzt ein eigenes verteiltes Dateisystem namens Disco Distributed File System (DDFS), dieses setzt auf dem jeweiligen systemnativen Dateisystem auf. Es ist komplett schemafrei, Daten werden als Blobs gespeichert, welche über Tags referenziert werden. Diese Tags können neben beliebig vielen Blobs auch andere Tags referenzieren, so wie benutzerdefinierte Metainformationen beinhalten. Daten werden über eine frei definierbare Anzahl an Knoten repliziert, um die Auswirkungen von Netzfragmentierung abzumildern und einen Grad an Ausfallsicherheit herzustellen. Netztopologische Eigenschaften (Abstand der Knoten zueinander) werden jedoch nicht berücksichtigt. Neue Knoten können zur Laufzeit über ein Webinterface hinzugefügt werden. Das Disco Projekt stellt wahlweise auch ein optionales Framework namens DiscoDB bereit. Dieses dient dem Speichern von (Schlüssel, Wert)-Paaren, ähnlich Python Dict. Die Datenaufrufe sind auf Geschwindigkeit optimiert. DiscoDB bedient sich zu diesem Zwecke CMPH für perfektes Hashing und Memory Mapping.
Die Dokumentation des Disco-Projektes entspricht den Erwartungen an ein Open-Source-Projekt. Sie ist einigermaßen ausführlich und verständlich geschrieben. Auf der Projektwebsite finden sich darüber hinaus einige Tutorials und eine technische Dokumentation. Ein offizielles Communityforum ist nicht vorhanden, es existieren jedoch eine Mailingliste und ein IRC-Channel. Nokia Disco steht sowohl als Quelltext, als auch als Debian Package bereit. Auf der Projektwebsite findet sich eine knappe Installationsanleitung, sowie ein Leitfaden zur Fehlerbehebung. Offiziell unterstützt werden alle Unix-basierten Betriebssysteme, inklusive MacOS.

Besonderheiten
Nokia Disco bietet laut Projektwebsite volle Interoperabilität mit Amazon EC2[DIS].

3.3.5 Spark

Spark ist ein Clustercomputing-System, welches an der Universität Berkeley unter der BSD-Lizenz entwickelt wird*[SPA]*. Die ursprünglich geplanten Einsatzgebiete waren iterative Algorithmen und interaktives Data Mining. Ziel der Entwicklung war eine Leistungssteigerung gegenüber Apache *Hadoop* in Hinsicht auf diese Einsatzgebiete. Da *Spark* auf *Mesos*, dem Cluster Manager des *Hadoop*-Projektes aufsetzt, ist eine gleichzeitige Verwendung von *Spark* und *Hadoop*, sowie der Zugriff auf von *Hadoop* erzeugte Datenquellen möglich. Neben einigen Forschungsgruppen an der Universität Berkeley wird Spark von einigen IT-Firmen (Klout, Quantifind) verwendet.
Das System verwendet eine eigene MapReduce-Implementierung, die laut *Spark*[SPA] bis zu 30x schneller als *Hadoop* operieren soll*[SPA]*. Diese Geschwindigkeit wird unter anderem durch die Programmiersprache Scala zur Ausführung von Aufträgen realisiert. Durch diese lassen sich MapReduce-Aufträge ohne große Umwege erstellen. Spark bietet eine große Menge an Analyse- und Monitoringwerkzeugen, welche das gesamte Funktionalitätsspektrum abdecken. Während *Spark* auch auf einem einzigen Rechner lauffähig ist, ist für die Nutzung in einem Cluster *Apaches* Cluster Manager *Mesos*[SPA] notwendig. Dieses bildet eine Schnittstelle zwischen *Spark* und einem *Hadoop* Cluster und bietet fehlertolerante Replikation eines Masterknotens, skaliert auf zehntausende Knoten, verteilt Rechenlasten und bietet Schnittstellen für weitere Programmiersprachen wie Java, Python und C++. Die weiteren dateisystemspezifischen Funktionalitäten entsprechen denen von Apache Hadoop *HDFS*. Eine webbasierte graphische Oberfläche erlaubt Einsicht in den Clusterzustand.
Grundlegende Informationen zur Installation und Nutzung von *Spark* sind auf ihrer Webseite und auf Github vorhanden. Erweiterte Informationen sind auf der IBM Homepage auffindbar. *Spark* kann mittels Git von Github heruntergeladen werden und erfordert Scala in der Version 2.8 oder 2.9. Für die Verwendung in einem Cluster ist zusätzlich Apache *Mesos* erforderlich. Eine Anleitung inkl. Konfigurationsskript für die Nutzung auf einem Amazon *EC2 Cluster* ist auch vorhanden. Als Betriebssystem wird Linux vorausgesetzt.

Besonderheiten
Die laut Herstellerbenchmark immense Geschwindigkeit von Spark ist hauptsächlich auf die Vermeidung der Festplattennutzung und auf die Nutzung des Arbeitsspeichers und Prozessorcaches zurückzuführen. Das heißt, alle Daten werden stets entweder von diesen schnellen Speichern gelesen oder auf diese geschrieben. Optional können Daten auch auf der Festplatte zwischengelagert werden. Durch die Lagerung im Arbeitsspeicher können Daten konstant wiederverwendet werden und müssen nicht erneut angefragt werden, dadurch eignet sich Spark besonders für sich wiederholende Aufgaben im Datenanalysebereich.

3.3.6 Myspace Qizmt

Qizmt ist ein auf C# basierendes MapReduce-Framework für Windows-Plattformen aus dem Hause MySpace*[QIZ]*. Ursprünglich für den internen Betrieb von MySpace entwickelt, wurde es 2009 als Open-Source-Projekt veröffentlicht. *Qizmt* besitzt ein eigenes MapReduce-Verfahren, welches Aufträge in C#.Net empfängt und unter anderem Datensammlung, -analyse und auch Medienverarbeitung ermöglicht. MapReduce-Aufträge können in einer eigenen Entwicklungsumgebung samt Debugger angeblich [QIZ] sehr schnell und einfach erstellt und von jedem Rechner aus im Cluster editiert, gestartet und administriert werden. Die Geschwindigkeit kann entsprechend des MapReduce-Modus (Sorted, Grouped, Hashsorted) flexibel verändert werden und soll laut Myspace [QIZ] insgesamt konkurrenzfähig sein. Für die weitere Betrachtung

ist die Tatsache interessant, dass *Qizmt* eine eigene Entwicklungsumgebung für Aufträge, zahlreiche Analysewerkzeuge und einen dedizierten Debugger für MapReduce-Aufträge bietet. Durch die mitgelieferte Entwicklungsumgebung und den Debugger ist das Erstellen von MapReduce-Aufträge unkompliziert und ohne größere Vorarbeit möglich. Die Steuerung des Clusters ist dezentral, das heißt es ist möglich, die Ausführung von Aufträgen auf jedem Knoten zu bearbeiten.
Qizmt besitzt ein eigenes Dateisystem namens *MR.DFS* und erzeugt bei Einrichtung auf mehreren Rechnern einen Windows-Cluster, welcher konfigurierbare Datenredundanz und Ausfallsicherheit bietet. Es hebt sich hierdurch nicht von den Erwartungen an eine verteiltes Dateisystem ab. Durch Hinzufügen von Rechnern kann laut *Myspace[QIZ]* sowohl die Speicherkapazität, als auch die Rechengeschwindigkeit erhöht werden. Ein Masterknoten wird nicht verwendet. Daten werden allgemein in einer eigenen Datenbank als Schlüssel und Werte gespeichert und zusätzlich indiziert. Positiv fällt auf, dass Aktivität und Fehlerfreiheit einzelner Clusterknoten und der aktuelle Auftragsstatus dazu innerhalb der Konsole überwacht werden können.
Eine sehr ausführliche, bebilderte und mit Beispielen versehene Dokumentation zur Installation, Nutzung und Fehlerbehebung von *Qizmt* kann auf der Homepage gefunden werden.
Ein Installer für die Nutzung auf einem Rechner oder in einem Windows-Cluster kann direkt von der Homepage heruntergeladen werden. Voraussetzungen sind entweder Windows Vista/7 oder Windows Server 2003/2008 mit .Net 3.5 SP1.

Besonderheiten
Man kann *Qizmt* auch mit Amazons EC2 – Clustern verwenden. Ein ausführliches Tutorial dafür wird auf der Homepage ebenfalls angeboten[QIT].

3.4 Implementierungen mit MapReduce ähnlichen Programmiermodellen

3.4.1 Gearman

Gearman ist ein leichtgewichtiges Open-Source-Anwendungsframework *[GRM]*. Es regelt die Distribution von Arbeitsvolumina an eine frei skalierbare Anzahl von Arbeiterinstanzen (*Worker*). Nicht nur bietet es APIs für eine Vielzahl gängiger Programmiersprachen, auch der Jobserver, der Kern des Frameworks, ist in drei verschiedenen Implementierungen erhältlich. Diese unterscheiden sich jedoch nicht bezüglich ihrer Funktionalität.
Das Framework bietet von sich aus keine MapReduce-Funktionalität. Operationen gemäß dem MapReduce-Paradigma lassen sich jedoch mit nur geringem Aufwand ausführen, da die Master/Worker-Architektur sich hierfür anbietet. Dies hebt Gearman in Bezug auf eine weitergehende Betrachtung von anderen Implementierungen ab. Besonders die Fähigkeit von Workerinstanzen, übertragene Arbeitslasten weiter zu delegieren, eignet sich hervorragend für eine Aufteilung in Map- und Reduceschritte. Darüber hinaus bieten einige der unterstützten Programmiersprachen (zum Beispiel PHP) bereits vorgefertigte Map- und Reducefunktionen. Da der Jobserver neue Tasks nur an Workerinstanzen im Idlemodus verteilt, ergibt sich für etwaige MapReduce-Anwendungen auch eine primitive Lastbalancierung. Da Gearman ein reines Kommunikationsframework darstellt, besitzt es weder eine eigene Datenbank, noch ein eigenes Dateisystem. Ausschließlich ein eigenes TCP-Protokoll für die Kommunikation mit dem Jobserver ist gegeben. Diese Tatsache macht Gearman, zusammen mit der Tatsache, dass es mehrere Implementierungen des Jobservers gibt
Wie die Mehrzahl der Open-Source-Projekte ist *Gearman* sehr gut dokumentiert. Auf der

Projektwebsite finden sich sowohl Einsteigertutorials, als auch technische Spezifikationen für Entwickler. Der *Gearman* Jobserver namens *gearmand* ist in drei verschiedenen Ausführungen erhältlich: Java, Perl und C/C++. Die Installationsanleitungen sind jedoch bei letzteren ein wenig dürftig und setzen einiges an technischem Fachwissen so wie Experimentierfreude voraus.
Die C/C++ Variante lässt sich (zumindest unter Linux) jedoch relativ leicht kompilieren und installieren[GRM].

Besonderheiten
Maximale Flexibilität wird durch eine Vielzahl unterstützter Programmiersprachen, wie C, Perl, PHP, Python, C#, JMS und Java gewährleistet. Zusätzlich dazu lässt sich die MapReduce-Funktionalität sehr einfach an die jeweiligen Projekterfordernisse anpassen. Clusterknoten können auf unterschiedlichen Betriebssystemen laufen und trotzdem miteinander interagieren und als ein Cluster erscheinen.

3.4.2 Stratosphere

Stratosphere wird von der Deutschen Förderungsgemeinschaft (DFG) gefördert und ist das Resultat eines Forschungsprojektes der technischen Universität Berlin, der Humboldt Universität zu Berlin und des Hasso-Plattner-Instituts in Potsdam*[SRA]*. Das Projekt verwendet das *PACT* Programmiermodell, welches eine betrachtenswerte Generalisierung und Erweiterung des MapReduce-Verfahrens darstellt, beispielsweise mit zusätzlichen Funktionen zweiter Ordnung[ADV10]. Die Ausführung der Aufgaben wird von der parallel entwickelten *Nephele Engine* übernommen. Gestartete *PACT*-Programme können per Log-Datei des JobManagers überwacht werden. Die Architektur ist ähnlich dem des Standard-MapReduce, mit einem JobManager als zentraler Komponente der Kommunikation zwischen den Clients, welcher sich um das Verteilen der Aufträge kümmert und überprüft, ob diese korrekt ausgeführt werden.
Stratosphere verwendet das *Hadoop Distributed Filesystem* (HDFS). Dieses kann besonders große Daten auf großen Clustern verteilt speichern und bietet unter anderem Fehlertoleranz, Skalierbarkeit und kann leicht erweitert werden. Bei *Stratosphere* muss das Hadoop-Verzeichnis auf jedem System an der gleichen Stelle zugreifbar sein, was dessen Flexibilität einschränkt.
Stratosphere bietet eine ausführliche Onlinedokumentation die sowohl Setup und Konfiguration der Cluster wie auch das Erstellen von *PACT*-Programmen inklusive Beispielen und Testfällen umfasst. Weiterhin werden eine Suchfunktion, diverse FAQs und eine Konfigurationsübersicht angeboten. Auf der Homepage finden sich die Binärpakete der Software in der aktuellen Version sowie Javadocs zum Download. In der Onlinedokumentation wird die Installation und das Einrichten der Cluster Schritt für Schritt erklärt. Nach aktuellem Stand wurde die Software auf Ubuntu, Debian und Suse Linux Systemen getestet. Für das Starten der Software wird außerdem eine Java 1.6 kompatible JRE benötigt.

Besonderheiten
Die eigens entwickelte *Nephele Engine* kümmert sich um die Ausführung der *PACT*-Programme. Aufträge die von JobManager verteilt werden, laufen in Instanzen auf den Clients, welche wiederum eine lokale Komponente des Nephele Frameworks bilden.
Außerdem wird anstelle von MapReduce das *PACT*-Modell verwendet. Dieses soll den Programmierern ein effizienteres Lösen von speziellen Problemen ermöglichen. Datenanalyse-Aufgaben können mit Straight-Forward-Datenflüssen beschrieben werden. Bei Parallelisierungen von Aufgaben werden dem ausführenden System mehr Freiheiten bezüglich der Programmstruktur ermöglicht als bei vergleichbaren MapReduce-Prozessen [SRA].

3.4.3 GridGain

GridGain ist eine in Java implementierte Open-Source-Middleware für Datenverarbeitung und -analyse *[GRG]*. Es bildet mit Schnittstellen zu unter anderem SQL und Hadoop HDFS eine Anwendungsschicht zwischen einem Data Warehouse und lokalen Anwendungen. Dabei ist die Software besonders interessant, wenn man mit sehr großen Datenmengen arbeitet. Beim Parallelisieren der Verarbeitungsprozesse kann auch auf MapReduce-Funktionalität zugegriffen werden. Dabei werden APIs für Java, Groovy und Scala zu Verfügung gestellt. *GridGain* stellt eine eigene MapReduce-Implementierung zu Verfügung. In der Dokumentation wird darauf hingewiesen, dass diese im Weiteren als MapReduce bezeichnet wird, diese jedoch wenig mit der von Google patentierten MapReduce Implementierung zu tun habe. Allerdings sei das Grundkonzept das gleiche und das Resultat vergleichbar, wodurch man die beiden Lösungen durchaus nebeneinander Stellen kann.
Die Dokumentation wirbt mit der hohen Performanz und Skalierbarkeit der Software. Im Gegensatz zu vielen Konkurrenten wird kein verteiltes Dateisystem angeboten. Stattdessen werden Daten konstant im Speicher gehalten um eine hohe Verfügbarkeit und schnellen Zugriff zu ermöglichen. Dieses Konzept wird in *GridGain* als In-Memory DataGrid bezeichnet. Diesen Schritt gehen die Entwickler auf Grund der Erfahrung mit langsameren verteilten Datensystemen mit Flaschenhals-Potenzial, welche bei realen Echtzeitberechnungen scheitern würden.
Vorhanden sind eine Online-Dokumentation, Erläuterungen zu möglichen Konfigurationen sowie viele Implementierungsbeispiele. Ein sehr ausführliches Online-Buch klärt nicht nur über die Benutzung und Funktionalität der Software auf, sondern bietet auch ein breites Wissen über Real Time Big Data Processing sowie die Nutzung von alternativen APIs an. Einige Videos führen durch Installation, Integration und erste Versuche mit *GridGain*. Außerdem wird ein Online-Forum angeboten, das zum Zeitpunkt des Verfassens dieses Dokuments gut und aktiv besucht wurde. Von *GridGain* existieren zwei verschiedene Versionen. Eine Community Edition (GPLv3) sowie eine kostenpflichtige Enterprise Edition. Als Betriebssysteme werden Windows XP/Vista/2007 , Linux/Unix (Ubuntu, Fedora) sowie Mac OS unterstützt. Benötigt zur Lauffähigkeit wird außerdem Java ab Version 6. Die Programmiersprachen Scala und Groovy sind optional und müssen gegebenenfalls noch zusätzlich installiert werden. Zum Installieren von *GridGain* muss laut Anleitung lediglich das ZIP-Archiv entpackt werden. Ein Installationsvideo sowie eine ausführliche Anleitung zur Konfiguration stehen in der Dokumentation zu Verfügung.

Besonderheiten
GridGain bietet als mögliche Job-Implementierungssprachen Java, Scala sowie Groovy an. Im Gegensatz zu Apache Hadoop werden nur die Berechnungen an die Worker verteilt, *Gridgain* bietet kein verteiltes Dateisystem. Stattdessen werden Daten im Speicher gehalten. Außerdem sind die Resultate der MapReduce-Aufträge auf Single-Values beschränkt.

3.4.4 Sector/Sphere

Sphere ist eine vom amerikanischen Center for Data Mining entwickelte parallele Prozessengine*[SEC]*. Sie baut auf dem verteilten Dateisystem *Sector* auf, welche vom selben Institut stammt. Laut einem Benchmark der Entwickler ist *Sector/Sphere* bei MapReduce-Aufträge 2 bis 4 mal schneller als Apache Hadoop[SEC]. Sowohl Sector als auch Sphere sind Open-Source.
Sector/Sphere nutzt ein unorthodoxes Modell der parallelen Prozessbearbeitung. Daten werden als so genannte Streams (Metadatei auf einem Sector-Dateisystem) an einen Sphere-Masterknoten

übergeben, welcher den Stream an Slaveknoten verteilt. Hierzu wird dem Masterknoten eine Indexdatei übergeben, welche die Aufteilung des Datenstroms in verschiedene Chunks beinhaltet. Diese Aufteilung in Chunks ist komplett vom Benutzer definierbar und ist völlig unabhängig von der Dateistruktur auf dem Sector-System. Die Programmierung von Sphere-Slaves geschieht etwas umständlich über vorkompilierte dynamische Bibliotheken, welche zur Laufzeit vom Hauptprogramm auf die Slaves hochgeladen werden. Der Output des Arbeitsauftrags wird wiederum als *Sphere*-Stream zurückgegeben, dessen weitere Verarbeitung auf Sector vom Benutzer spezifiziert werden kann. Hierbei ist vor allem der *Bucket Output* interessant, da er Listen von Ergebnissen nach einem Schlüssel gruppiert zurück gibt, und somit den Reduce-Teil des MapReduce-Paradigmas implementiert. Sector bietet eine überdurchschnittlich große Menge an Monitoringwerkzeugen, inklusive grafischer Benutzerschnittstellen.
Sector ist ein verteiltes Dateisystem, welches besonders für leseintensive Einsatzszenarien geeignet ist. Einzelne Dateien werden, nach dem Upload auf einen austauschbaren Masterknoten, auf mehrere netztopologisch möglichst weit von einander entfernte Knoten repliziert. Während des Lesevorgangs organisiert dann der Masterknoten, ähnlich einem *Peer-to-Peer*-System, eine simultane Übertragung der Daten an den *Sphere*-Client. Sector unterstützt für die Verwendung mit Applikationen außer Sphere die gängige Dateisystemsemantik (Dateipfade/Ordner) und gängige APIs für den Datenzugriff (mkdir, rm, open, read etc.)
Die Dokumentation zu Sphere ist verständlich, jedoch lässt der Umfang stark zu wünschen übrig. Sie verweist zum Großteil auf Beispieldateien, welche mit dem Quelltext ausgeliefert werden. Allerdings ist die SourceForge Community vom Projekt zum Zeitpunkt dieser Studie sehr aktiv und gibt brauchbare Hilfestellung. Sector/Sphere ist nur als Tarball erhältlich, eine Installationsdokumentation konnte zum Zeitpunkt der Recherche nicht aufgespürt werden.

Besonderheiten
Ein neuartiges Dateisystem namens Grid Computing wird angeboten und Aufträge werden als vorkompilierte Bibliotheken anstatt als Skripte gehandhabt.

3.5 Übersicht der Bewertungsergebnisse

Im Folgenden können alle Implementierungen, ihre Teilpunkte und das Gesamtergebnis in einer Tabelle wie in *Abbildung 3.5.1* (vgl. auch *Anhang 2.1* bzw. *Anhang 2.2*) begutachtet werden.

	Apache Hadoop	MapR	Cloudera	Cascading	Basho Riak	Aster Data SQL-MapReduce	Mongo DB	Nokia Disco	Spark	MySpace Qizmt	Gearman	Stratos phere	Gridgain	Sector/ Sphere
MapReduce-Implementierung														
Vorhandensein	0	0	0	0	0	0	0	0	-	0	-	-	0	0
Flexibilität	+	+	+	+	+	+	-	0	+	0	+	0	+	-
Skalierbarkeit	0	+	+	0	0	+	0	+	0	+	0	0	0	0
Einfachheit	+	+	0	+	+	+	0	-	+	+	0	+	+	0
Boni (max 6)	+	+++	++++	+	+++	+++	+++	++++	++++	++++	+++	++++	+++	++++
Datenbank/Dateisystem														
Vorhandensein	0	0	0	0	0	0	-	0	0	0	-	0	-	0
Skalierbarkeit	0	0	0	0	+	0	0	0	+	0	0	0	0	0
Sicherheit	0	+	+	0	+	-	+	0	+	+	-	-	-	+
Einfachheit	-	+	+	-	+	+	+	+	0	+	0	0	+	0
Boni (max 3)	+	+++	+	+	++	+	+	+	++	+		++	++	++
Dokumentation														
Vorhandensein	0	0	0	0	0	*	0	0	0	0	0	0	0	0
Ausführlichkeit	+	+	+	+	+		+	0	0	0	+	0	+	-
Komfort	0	0	-	-	0		0	0	-	+	0	+	+	0
Boni (max 2)	++	+++	+	+	+++		++	+		+	++	+	++	+
Installation														
Installationsanleitung	0	0	+	0	0		0	0	0	0	0	0	0	0
Betriebssyteme	0	+	0	0	+		+	0	0	0	-	0	+	-
Boni (max 1)	+	+	+		+		+			+	+	+	+	0
Besonderheiten	+++	++	++	+	++	++	+	+	+++	+	+++	++++	++++	++
Endergebnis	+10	+19	+14	+5	+18	+10	+10	+8	+11	+13	+7	+12	+15	+7

Abbildung 3.5.1: Übersicht der Bewertungsergebnisse

Wie sich aus der Tabelle ableiten lässt, schwanken die Punktzahlen sowohl unter den Kandidaten, als auch innerhalb der Kategorien teilweise deutlich. Da grundlegende Funktionalitäten nur mit einer Null bewertet wurden und hervorzuhebende Eigenschaften einen zusätzliches Plus erhielten, ist eine Endbewertung mit 5 von maximal möglichen 32 Punkten kein Indiz für eine schlechte Implementierung.
Alle hier aufgeführten Implementierungen erfüllen ein Mindestmaß an soweit möglich abschätzbarer Nutzungsqualität. Dennoch können jetzt umso klarer die Unterschiede im zusätzlichen Umfang des jeweiligen Kandidaten ersehen werden. Als nächstes möchten wir die näher zu betrachtenden Implementierungen aufzählen und begründen, warum wir diese und nicht andere ausgewählt haben.

3.6 Die näher zu betrachtenden 6 Kandidaten

Nach einem umfangreichen Feature-Vergleich und einer anschließenden Punktebewertung der Implementierungen, konnten sich einige von ihnen besonders hervorheben und scheinen für eine konkretere Betrachtung interessant zu sein. Die höchste Gesamtpunktzahl ist allerdings nicht von vornherein ein Eignungsgarant, wenn diese Implementierung zum Beispiel nicht innerhalb einer vernünftigen Zeit installier- und konfigurierbar ist. Durch seine weltweite Bekanntheit und Verwendung in vielen Unternehmen bietet sich als Referenzimplementierung *Apache Hadoop* an. Alle weiteren Implementierungen werden zwar auch untereinander, aber besonders in Bezug auf *Apache Hadoop* getestet werden.

Des weiteren hat unter den *Hadoop–basierenden Implementierungen MapR* eine hohe Gesamtwertung erreicht. Die Distribution hat aufgrund der überlegenen Dokumentation und den komfortablen Clusterzugriffseigenschaften wie Direct Access NFS (Network File System) eine höhere Bewertung als der Hauptkonkurrent Cloudera erhalten und erhält daher den Vorzug. Auch wenn manche Sicherheitseigenschaften wie Datenspiegelung, Snapshots und der 24-Stunden-Entwicklersupport nur in der kostenpflichtigen Version von MapR enthalten sind, kann diese dennoch 30 Tage lang getestet werden. In Anbetracht dessen, dass die freie M3-Version Apaches Hadoop, Hive, Pig und HBase vollständig mit einbezieht, um eine grafische Bedienoberfläche, Management-und Überwachungsfunktionen und um DA-NFS erweitert und darüber hinaus für professionellere Ansprüche auch als kommerzielle Version existiert, ist dieser Kandidat besonders interessant.
Unter den *nicht-Hadoop-basierenden Implementierungen* bietet sich als erstes *Basho Riak* an. Es hebt sich besonders durch seine Large-Scale-Analysemöglichkeiten hervor und ist auch insgesamt ein sehr starker Kandidat, der ein breites Spektrum an interessanten Features aufweist. Beispiele hierfür sind der flexible Einsatz von Erlang oder Javascript für MapReduce-Aufträge, die vereinfachte Steuerung der Aufträge über eine HTTP-Schnittstelle, sowie die umfangreiche grafische Bedienoberfläche Riak Control. Sicherheitsfeatures wie Spiegelungen und Wiederherstellungsoptionen gehören im Gegensatz zur freien *MapR*-Version ebenso zur Grundausstattung. Letztendlich führt die, im Vergleich zu *Aster Data, Nokia Disco* und *Spark*, sehr gute Dokumentation und Installationsanleitung dazu, dass Riak für eine genauere Betrachtung vorgezogen wird.
Als nächstes sticht in dieser Kategorie *MySpace Qizmt* hervor, das neben der relativ zu den anderen Kandidaten hohen Gesamtpunktzahl gute Installations- und Bedieneigenschaften aufweist. *MongoDB* ist umfangreicher dokumentiert , dafür bietet *Qizmt* eine eigene Entwicklungsumgebung samt Debugger und erlaubt eine einfache und unterstützte Erstellung von MapReduce-Aufträgen. Obwohl *Spark* enorme Geschwindigkeit verspricht, in dem es nur innerhalb des Prozessorcaches und Hauptspeichers operiert, spielt dieser Faktor in Anbetracht der Unmöglichkeit eines aussagekräftigen Tests der Implementierungsgeschwindigkeiten keine Rolle. Aster Data konnte uns keine kostenlose Testversion ihrer Implementierung zur Verfügung stellen und *Nokias Disco* ließ sich nicht ohne Weiteres auf unseren Testsystemen installieren, womit *Qizmt* den rundesten Eindruck erweckt.
Aus der Kategorie der *MapReduce-ähnlichen Implementierungen* ist *GridGain die* gesamtpunktstärkste Implementierung. Mit sehr guter Dokumentation und sehr ausgeprägten Installationsmöglichkeiten hebt sie sich von Sector/Sphere und Gearman deutlich ab. Letztere Implementierung erfordert mangels einer integrierten Datenhaltung teils tiefer gehendes technisches Server-Know-How um überhaupt in Betrieb genommen werden zu können. Aufgrund der vielen Analyse- und Monitoringtools und der Möglichkeit MapReduce-Auträge in Echtzeit ändern zu können, bietet sich *GridGain* gut für weitergehende Betrachtungen an.
Die nächsthöchste Bewertung fällt auf *Stratosphere*. Obwohl dessen PACT-Modell vielversprechende und einzigartige Eigenschaften besitzt, mangelt es dem Projekt gleichzeitig an wichtigen funktionalen Anforderungen. Unter anderem betonen die Projektverantwortlichen, dass ihre Implementierung zum momentanen Zeitpunkt (2012) keine Toleranz bezüglich Fehlverhalten in MapReduce-Aufträgen und bei Datenverlust bietet. Darüber hinaus ist seit mehreren Wochen (Stand 24.05.2012) ihre Hauptdomain im Internet nicht mehr verfügbar und lässt an deren langfristige Existenz und Support zweifeln. Interne Vortests haben gezeigt, dass stattdessen *MongoDB* ein interessanterer Kandidat wäre. Zum einen ist es, verglichen mit Spark, sehr einfach einzurichten und besonders gut dokumentiert, zum anderen unterscheidet es sich von den

anderen Kandidaten insgesamt durch seine Datenbankbasis mit vollständigem Index Support, Replikation und hoher Verfügbarkeit. Wiederholte MapReduce-Aufträge berechnen nur neu hinzugekommene Daten und integrieren die Ergebnisse in bereits bestehende, um Ressourcen und Zeit zu sparen. Darüber hinaus wird *MongoDB* von großen Unternehmen wie SAP, und Disney und hunderten Seiten wie Sourceforge und IGN in der Produktion verwendet (Quelle siehe Anhang 1.3 , Spalte Popularity). In Anbetracht der genannten Nachteile, die *Sector/Sphere*, *Gearman*, *Spark* und *Disco* aufweisen und *Cascading* von vorneherein aufgrund seiner mäßigen Leistung in vielen Bereichen ausgeschlossen wurde, bleibt als naheliegendste Implementierung *MongoDB*.

4 Detaillierte Betrachtung ausgewählter Implementierungen

4.1 Kriterien und Parameter für die detaillierte Betrachtung

Um die ausgewählten Kandidaten sinnvoll vergleichen zu können, muss zuerst die Schnittmenge gleicher oder sehr ähnlicher Funktionen und Eigenschaften ermittelt werden. Als zweites müssen die Vergleichsparameter definiert werden. Als erste Orientierungsbasis dient das Kapitel *Vorauswahl relevanter MapReduce-Implementierungen.* Dieses unterteilt jede Implementierung in:

1. **MapReduce-Implementierung**
2. **Datenbank/Dateisystem**
3. **Dokumentation**
4. **Installation**
5. **Besonderheiten**

Die Punkte *Installation, Dokumentation* und *Besonderheiten* entfallen. Die Punkte *Installation* und *Dokumentation* aufgrund der Tatsache, dass im produktiven Betrieb der zusätzliche Aufwand durch eine schwierige, beziehungsweise schlecht dokumentierte Installation vernachlässigbar ist. Der Punkt *Besonderheiten* entfällt, da diese meist direkte Auswirkungen auf Vergleichskriterien der anderen Punkte haben, und somit durch diese ausreichend beschrieben sind. Sollten doch Besonderheiten existieren, welche nicht durch die vorhandenen Vergleichskriterien abgedeckt sind, so werden diese separat getestet. Der Punkt Datenbank/Dateisystem wird durch den Punkt Infrastruktur ersetzt, um den Aspekt der Clustersteuerung mit zu umfassen. Somit bleiben die in der folgenden Tabelle aufgelisteten Aspekte für einen detaillierten Test der ausgesuchten Implementierungen:

4.1.1 Kriterien und Vergleichsaspekte

MapReduce-Implementierung

Vergleichskriterium	Testsfälle/praktische Aspekte / selbstbewertet	Theoretische / nichtfunktionale Aspekte / informativ
1. Erstellung von MapReduce-Funktionen	1.1 Zählen des Vorkommens einzelner Wörter in einem Dokumentsatz (klassisches MapReduce Beispiel)	1.3 Unterstützte Sprachen bei der Erstellung von MapReduce Funktionen 1.4 Wie werden MapReduce-Aufträgen erstellt bzw. geändert (Konsole/GUI)
2. Ausführen von MapReduce-Aufträgen	2.1 Nötige Vorarbeit (Kompilieren von Skripten, Konfiguration von Knoten, etc.) 2.2 Schnittstelle zur Job-Queue 2.3 Möglichkeiten zur Weiterverwendung der Ergebnisse	2.4 Kompatibilität von Knoten bei unterschiedlicher Hardware/Betriebssystem

3. Skalierung	3.1 Hinzufügen eines Knotens 3.2 Hinzufügen eines Knotens zur Laufzeit 3.3 Entfernen eines Knotens 3.4 Entfernen eines Knotens zur Laufzeit	3.5 Skalierbarkeit auf hohe Anzahl von Knoten 3.6 Automatische Skalierbarkeit
4. Lastbalancierung		4.1 Vorhandensein 4.2 Konfigurationsmöglichkeiten bei automatischer Lastbalancierung

Infrastruktur

Vergleichskriterium	Testsfälle/praktische Aspekte	Theorie / nichtfunktionale Aspekte
5. Dateisystem	5.1 Einlesen und Konvertieren von Daten aus dem systemnativen Dateisystem 5.2 Export von Daten in das systemnative Dateisystem	5.3 Unterstützte Dateisysteme 5.4 Zugriffsmöglichkeiten auf das Dateisystem 5.5 Datenkonsistenz bei Ausfall eines Knotens 5.6 Vorteile einer größeren Knotenanzahl
6. Fehlertoleranz	6.1 Verhalten bei Ausfall eines Knotens testen	6.2 Konfigurationsmöglichkeiten / Verhalten bei Ausfällen von Knoten 6.3 Verhalten von kurzzeitigem Verbindungsverlust zu einem Knoten
7. Monitoring	7.1 Aktive MapReduce-Arbeitsaufträge 7.2 MapReduce-Arbeitsaufträge in der Queue 7.3 Verfügbarkeit von Knoten 7.4 Arbeitslast von Knoten 7.5 Aggregierte Sichten (Mehrere Knoten, mehrere Werte etc.) 7.6 Existenz/Inhalt von Logfiles	7.7 Welche Oberflächen/GUI werden angeboten 7.8 Welche Monitoringmöglichkeiten gibt es

Besonderheiten

Weiteres Kriterium für die Betrachtung sind eventuelle Besonderheiten der jeweiligen Implementierungen. Dies könnten zum Beispiel Erweiterungen des MapReduce-Konzepts, eigene Testframeworks oder unkonventionelle Dateisysteme sein. Es wäre in vielen Fällen möglich, diese Besonderheiten einem der oben genannten Vergleichskriterien zuzuordnen. In solchen Fällen, in denen dies Sinn ergibt, das heißt, die Besonderheiten der Implementierung lassen sich sinnvoll in die Fragestellung eines Testaspektes einordnen, wird die Besonderheit auch an diesem Punkt angesprochen. Der Übersicht halber werden diese Aspekte jedoch noch einmal separat aufgeführt.

4.1.2 Erläuterung der Fragestellung der Vergleichsaspekte

Erstellung von MapReduce-Funktionen

Das *Zählen des Vorkommens einzelner Wörter in einem Dokumentsatz* beschreibt ein einfaches Beispiel einer MapReduce-Funktion. Anhand von diesem wird der generelle Aufbau eines MapReduce-Aufträge in einer Implementierung betrachtet. Beim Aspekt *Unterstützte Sprachen bei der Erstellung von MapReduce Funktionen* handelt es sich um eine kurze Aufzählung der Programmier-/Skriptsprachen, mit denen unter der betrachteten Implementierung MapReduce-Aufträge erstellt werden können. Zusätzlich erfolgt eine kurze Analyse, inwiefern sich die Ausführung unter verschiedenen Sprachen unterscheidet. Hier ist zum Beispiel die Frage, ob Zusatzmodule verwendet werden müssen und inwiefern Einschränkungen existieren. Es ist für die Fachstudie von Interesse, wie *MapReduce*-Aufträge *erstellt/geändert* werden. Aus diesem Grund wird betrachtet, auf welche Weise sich Aufträge starten lassen (zum Beispiel SSH/Telnet auf Masterknoten, dedizierter Port, WebUI) und um welche Form von Benutzerschnittstelle es sich dabei handelt. Zusätzlich werden die Konfigurationsmöglichkeiten der Aufträge betrachtet, auch in Hinblick auf die zu übergebenden Parameter.

Ausführen von MapReduce-Aufträgen

Unter dem Aspekt *nötige Vorarbeit* wird betrachtet, welche Vorbereitungen seitens des Nutzers getroffen werden müssen, um einen MapReduce-Job auszuführen. Hierzu gehört das etwaige Vorkompilieren von Laufzeitbibliotheken oder JAR-Dateien, die Konfiguration der Knoten und die nötigen Schnittstellen, welche von Map- und Reduce-Funktionen implementiert werden müssen. Betrachtenswert ist auch die *Schnittstelle zur Job-Queue*. Dieser Punkt beschäftigt sich damit, wie MapReduce-Aufträge an die Job-Queue übergeben werden. Die wichtigsten Fragestellungen sind hier, ob sich die Ausführungsdateien auf dem Cluster befinden müssen, und ob sich die Funktionalität des Clusters in den Ablauf eines externen Programms einbinden lässt. Die *Möglichkeiten zur Weiterverwendung der Ergebnisse* stellen einen Interessanten Aspekt bei der Betrachtung der Implementierungen dar. Die Fragestellung bezieht sich hierbei hauptsächlich darauf, mit welchem Aufwand die Ausgabedaten eines MapReduce-Aufträge als Eingabedaten eines folgenden verwendet werden können. Weiter werden die hierbei zu berücksichtigenden Beschränkungen der Datentypen und die Identifikation der Daten auf dem Dateisystem betrachtet. Als letzter Punkt wird die *Kompatibilität von Knoten bei unterschiedlicher Hardware oder unterschiedlichem Betriebssystem* betrachtet.

Skalierung

Beim Aspekt *Hinzufügen eines Knotens* geht es darum, wie viel Konfigurationsaufwand für das Erweitern des Clusters um einen Knoten notwendig ist. Betrachtet werden sowohl der Konfigurationsaufwand am neuen Knoten selbst, als auch der am Rest des Clusters. Das *Hinzufügen eines Knotens zur Laufzeit* beschäftigt sich damit, inwiefern sich dieses vom Hinzufügen eines Knotens bei einem deaktivierten Cluster unterscheidet. Interessant beim *Entfernen eins Knotens* ist die Frage, welche Konfigurationsänderungen hierfür notwendig sind. Des weiteren werden eventuelle Auswirkungen auf die Datenkonsistenz eines verteilten Dateisystems betrachtet. Beim *Entfernen eines Knotens zur Laufzeit* werden die Auswirkungen auf eventuell noch auf dem Cluster laufende Arbeitsaufträge analysiert.

Lastbalancierung
Der Aspekt *Vorhandensein* befasst sich mit der Existenz vorgefertigter Mechanismen zum Lastausgleich in einer Implementierung. Es wird zwischen Automatismen und manuellen Konfigurationsmöglichkeiten unterschieden. *Konfigurationsmöglichkeiten bei automatischer Lastbalancierung* beziehen sich hier neben ihrem bloßen Vorhandensein auf ihren Umfang, so wie die Möglichkeit, diese Konfigurationsmöglichkeiten zur Laufzeit des Clusters zu ändern.

Dateisystem
Bei verteilten Dateisystemen ist die Frage interessant, wie das *Einlesen und Konvertieren von Daten aus dem systemnativen Dateisystem* umgesetzt wurde. Möglichkeiten wären hier zum Beispiel ein Drag-and-Drop-Verfahren, Standardsystemkommandos oder spezielle Konvertierungsprogramme. Zusätzlich dazu wird betrachtet, inwieweit die Datenstruktur der eingelesenen Dateien übernommen wird. Analog zu letzterem wird auch der *Export von Daten in das systemnative Dateisystem* betrachtet. Der Aspekt *Unterstützte Dateisysteme* bezieht sich sowohl auf die von der Clusterarchitektur unterstützten Dateisysteme, als auch auf jene, welche als Datenquellen für MapReduce-Aufträge verwendet werden können. Des weiteren wird betrachtet, ob ein eventuell vorhandenes eigenes Dateisystem auf dem systemnativen Dateisystem aufsetzt, oder ein eigenes Partitionsformat implementiert. Von Interesse sind auch die MapReduce-externen *Zugriffsmöglichkeiten auf das Dateisystem*. Betrachtet wird hier, ob sich das Dateisystem in das Wirtsbetriebssystem einhängen (*mounten*) lässt und über welche APIs darauf zugegriffen werden kann. Auch etwaige Methoden zum Betrachten der Daten (GUI, Konsolenprogramm), welche die betrachtete Implementierung beinhaltet, werden analysiert.
Beim Punkt der *Datenkonsistenz beim Ausfall eines Knotens* geht es um die Frage, ob beim Ausfall eines Knotens Datenverlust auftreten kann. Falls dies der Fall sein sollte, werden eventuell vorhandene Selbstheilungsmechanismen des verteilten Dateisystems betrachtet. Auch etwaige Geschwindigkeitseinbußen des Clusters sind von Interesse. Mögliche *Vorteile einer erhöhten Knotenanzahl* sind Geschwindigkeitszuwachs bei der Ausführung von MapReduce-Aufträge, besserer Schutz vor Datenverlust und der Umfang des Speicherzuwachses.

Fehlertoleranz
Der Aspekt *Verhalten bei Ausfall eines Knotens* befasst sich mit den Auswirkungen eines Knotenausfalls auf den Cluster. Betrachtet werden eventuell vorhandene *Single Points of Failure*, die Methode der Ausfallfeststellung, der Umfang des verlorengegangenen Arbeitsaufwandes, sowie die Existenz und Implementierung eines Checkpointing-Systems, um diesen zu verringern.
Das *Verhalten bei kurzzeitigem Verbindungsverlust zu einem Knoten* ist insofern interessant, als das möglicherweise zwischen temporären Verbindungsabbrüchen und einem kompletten Knotenausfall unterschieden werden kann. Zusätzlich dazu wird betrachtet, ob ein Knoten nach einem kurzzeitigen Verbindungsverlust automatisch in den Cluster reintegriert wird und was mit eventuell auf dem Knoten ausgeführten Tasks und deren Ergebnissen geschieht.

Monitoring
Das Vorhandensein folgender *Standardinformationen* wird untersucht:

- Aktive Aufträge,
- Aufträge in der Queue,
- Verfügbare Knoten,
- Arbeitslast der Knoten,
- Aggregierte Sichten.

Die *Logfiles* der betrachteten Clusterimplementierung werden nach Konfigurationsmöglichkeiten, Gliederung nach Clusterkomponente und Umfang der geloggten Ereignisse untersucht. Zusätzlich dazu werden die zur Verfügung gestellten *Oberflächen* betrachtet. Unterschieden wird hier zwischen einfachen, textbasierten Metriken, dedizierten Logreadern und grafischen HTTP-Benutzerschnittstellen.
Unter dem Aspekt *Welche Monitoringmöglichkeiten gibt es*, werden vor allem die vom Cluster zur Laufzeit oder nach Abschluss eines Auftrags erhobenen Metriken, ihr Umfang, so wie Möglichkeiten zur Definition eigener Metriken und Testfälle betrachtet. Ebenfalls von Interesse sind eventuelle Möglichkeiten, auf diese extern, zum Beispiel über eine HTTP-Schnittstelle oder Programme von Drittanbietern zuzugreifen.

4.2 Verwendete Testsysteme

Im Zuge dieser Fachstudie musste früher oder später darüber entschieden werden, mit welcher Hard- und Software und auf welche Art und Weise die einzelnen MapReduce-Implementierungen getestet werden sollen. Obwohl diese potenziell tausende von Knoten in einem Cluster unterstützen und unterschiedlichste Testszenarien möglich sind, musste aufgrund der Knappen Ressourcen der Verfasser ein Kompromiss eingegangen werden, der dennoch die Grundfunktionalitäten der Implementierungen berücksichtigt und zumindest deren Potenzial erahnen lässt. So entschieden wir uns je nach Möglichkeit entweder einen Zwei-Knoten-Cluster bestehend aus zwei physischen Rechnern aufzubauen, oder mehrere virtuelle Maschinen auf einem Rechner zu einem Pseudocluster zu verbinden. Um auch innerhalb einer vernünftigen Zeit interessante Testergebnisse vorweisen zu können, war es nötig die Tests parallel durchzuführen, daher konnte nicht jede Implementierung auf dieselbe Hardwareumgebung zurückgreifen. Da zudem nicht jede Implementierung auf demselben Betriebssystem lauffähig ist und nicht die gleichen Softwarekomponenten für einen Betrieb voraussetzt, wird in der folgenden Tabelle die Testumgebung für jede Implementierung angegeben.

Apache Hadoop			
Software		Hardware	
Version:	Hadoop 1.0.3 stable	CPU:	Intel C2Quad Q9500 @2,83 GHZ (4 Kerne / 4 Threads)
Betriebssystem:	Ubuntu 12.04 x64 auf Oracle VM VirtualBox, Gasterweiterungen installiert	RAM:	4 GB DDR3
Zusätzliche Software:	Oracle JDK 1.7 x64, OpenSSH 5.9	Cluster-typ:	Pseudocluster mit virtuellen Knoten
MapR			
Software		Hardware	
Version:	MapR 1.27 stable	CPU:	Intel Core i3 2100 @3,1 GHz (2 Kerne / 4 Threads)
Betriebssystem:	Ubuntu 12.04 x64 auf Oracle VM VirtualBox, Gasterweiterungen installiert	RAM:	8 GB DDR3
Zusätzliche Software:	Oracle JDK 1.7 x64	Cluster-typ:	Pseudocluster mit virtuellen Knoten
GridGain			
Software		Hardware	
Version:	GridGain 4.0.2 Community Version GridGain 4.1.1 Enterprise Version	CPU1: CPU2:	Intel Core2Duo 6420@ 2,13 GHZ (2 Kerne / 2 Threads) AMD Athlon X2 @ 2,0 GHZ (2 Kerne / 2 Threds)
Betriebssystem1: Betriebssystem2:	Windows 7 x64 Windows 7 x86	RAM1: RAM2:	6 GB DDR2 3 GB DDR2

Zusätzliche Software:	Oracle JRE 1.7 x32, Netbeans 7.1, Eclipse 4.2, optional scala-2.9.1-1, Groovy 1.8.6	Cluster-typ:	Cluster aus zwei physischen Knoten
Basho Riak			
Software		Hardware	
Version:	Riak 1.1.4 stable	CPU:	Intel Core i3 2100 @3,1 GHz (2 Kerne / 4 Threads)
Betriebssystem:	Ubuntu 12.04 x64	RAM:	8 GB DDR3
Zusätzliche Software:	curl 7.21.6, ncurses 5.9, openssl 1.0.1c, libssl 0.9.8, build-essential 1.0.2, libc6 2.13-34, Erlang 5.8.4 (r14b03), Rebar 2.0.0	Cluster-typ:	Pseudocluster mit virtuellen Knoten
MongoDB			
Software		Hardware	
Version:	MongoDB 2.0.6 stable	CPU:	Intel C2Quad Q9500 @2,83 GHZ (4 Kerne / 4 Threads)
Betriebssystem:	Windows 7 x64	RAM:	4 GB DDR3
Zusätzliche Software:	keine	Cluster-typ:	Pseudocluster mit virtuellen Knoten
Myspace Qizmt			
Software		Hardware	
Version:	Qizmt 1.2 Alpha	CPU1: CPU2:	Intel Core2Duo 6420 @2,13 GHZ (2 Kerne / 2 Threads) AMD Athlon X2 @ 2,0 GHZ (2 Kerne / 2 Threds)
Betriebssystem1: Betriebssystem2:	Windows 7 x64 Windows 7 x86	RAM1: RAM2:	6 GB DDR2 3 GB DDR2
Zusätzliche Software:	Oracle JRE 1.7 x64 / x86	Cluster-typ:	Cluster aus zwei physischen Knoten

4.3 Apache Hadoop

Apache Hadoop ist die wohl bekannteste Implementierung des MapReduce-Paradigmas. Die Architektur orientiert sich an der 2004 in einem Paper beschriebenen [DNS06]. Dabei besteht *Hadoop* aus fünf Teilen, welche als einzelne *Daemons,* Programm welche im Hintergrund ausgeführt werden. Der *JobTracker* übernimmt die Funktion des Masterknotens bezüglich MapReduce-Operationen, er verteilt MapReduce-Auftrag an die anderen Knoten, wo diese von den jeweiligen *TaskTrackern* empfangen werden. Das verteilte Dateisystem *HDFD* basiert auf einem ähnlichen Konzept: Die Metadaten des Dateisystems liegen auf einer zentralen *NameNode*, welche diese an die *DataNodes* auf den anderen Knoten weitergibt. Eine Sonderstellung besitzt die so genannte *Secondary NameNode*. Anders als der Name vermuten lassen würde, handelt es sich hierbei nicht um eine sekundäre *NameNode*, welche im Falle eines Ausfalls die Aufgaben der

NameNode übernimmt, sondern um den Checkpoint-Mechanismus von *HDFS*. Die *Secondary NameNode* fertigt hierfür Kopien der Daten der *NameNode* an, auf welche nach einem Neustart jener zurückgegriffen werden kann.

4.3.1 Installation, Inbetriebnahme und Dokumentation

Die *Installation* von *Apache Hadoop* war auf dem Testsystem ohne größere Probleme möglich, setzte jedoch einiges an Fachwissen voraus, da die Installationsanleitung an einigen kritischen Stellen Lücken aufweist (SSH-Konfiguration, Deaktivierung von IPv6). Seltsamerweise gab sich die getestete Version 1.0.3 nach der Kompilierung gegen die Testumgebung als Version 1.0.4 aus.
Die getesteten nicht in den *Hadoop*-Kern integrierten Erweiterungen sind zum Zeitpunkt der Fachstudie (*Hadoop* Version 1.0.3) in einem miserablen Zustand. Einige sind im Auslieferungszustand nicht kompilierfähig, und müssen manuell mittels eines speziellen Editors nachgepatcht werden (Gridmix), andere funktionieren nur eingeschränkt oder überhaupt nicht in der Testumgebung (Eclipse-Plugin, Capacity Scheduler).
Der *Konfigurationsaufwand* von *Apache Hadoop* ist relativ gering, lediglich einige XML-Dateien müssen zur Inbetriebnahme editiert werden. Die einzelnen Bestandteile von *Hadoop* (J*obTracker, TaskTracker, NameNode* etc.) lassen sich entweder einzeln, oder alle gemeinsam bequem über ein mitgeliefertes Skript starten. Das Feedback beim Start lässt jedoch zu wünschen übrig. Durch Konfigurationsfehler nicht startende Komponenten versagen still im Hintergrund und sind nur durch Analyse der jeweiligen Log-Dateien zu entdecken.
Die *Dokumentation* des *Hadoop*-Projektes wirkt auf den ersten Blick sehr umfassend und verständlich. Versucht man allerdings, konkrete Problemstellungen mit dieser zu lösen, erweist sie sich schnell als vollkommen unzureichend. Der Großteil der Dokumentation ist stark veraltet und bezieht sich auf seit längerer Zeit (bis zu einem Jahr) nicht mehr aktuelle Hadoop-Versionen. Zu dem werden auch in aktuelleren Abschnitten und Codebeispielen Klassen und Methoden verwendet, welche in der Codebase als obsolet (*deprecated*) gekennzeichnet sind. Darüber hinaus findet sich zu einer Vielzahl der auf der Website gepriesenen Erweiterungen überhaupt keine Dokumentation, der Anwender wird hier vollständig allein gelassen und ist gezwungen, das Internet nach Problemlösungen zu durchforsten.

4.3.2 Erstellung von MapReduce-Funktionen in Java

In diesem Abschnitt wird die Erstellung von MapReduce-Funktionen in Java betrachtet. Hierzu werden minimal drei Java-Klassen verwendet: *Job*, *Mapper* und *Reducer.* Apache Hadoop definiert MapReduce-Aufträge als Java-Objekte, entweder als Instanzen der Klasse *JobConf* oder der neuen Klasse *Job*. Obwohl die offizielle Dokumentation des *Hadoop*-Projektes ausschließlich Beispiele mit *JobConf*-Instanzen enthält, wird diese Klasse in derselben Dokumentation als überholt (*deprecated*) gelistet. Aus diesem Grund wird in diesem Dokument nur auf die Klasse *Job* eingegangen. Diese erlaubt über ihre Methoden, einen MapReduce-Auftrag zu konfigurieren, an den Cluster zu übergeben, die Ausführung zu steuern, und seinen Zustand abzufragen.
Zur Minimalkonfiguration eines *Job*-Objektes gehören:

- Ein- und Ausgabepfad,
- Datentypen der Ausgabe,
- die Klasse des Mappers,
- die Klasse des Reducers,
- IP/Hostname und Port des Jobtrackers.

In *Apache Hadoop* wird die Map-Funktion eines MapReduce-Aufträge als eigene Klasse definiert,

welche die vordefinierte Klasse *Mapper* beerbt. Herzstück dieser Klasse ist die Funktion *Map*, welche die eigentliche Map-Funktion implementiert. Die Funktion *Map* besitzt drei Parameter: key, value und context. Context dient hierbei als Zugriffszeiger auf das Mapper.Context-Objekt, welches unter anderem eine Instanz der Klasse *RecordWriter* beinhaltet, um auf HDFS schreiben zu können. Die Klasse *Reducer* ist in ihrer Benutzung analog zur Klasse *Mapper*. Die Funktion *Reduce* beinhaltet die Reduce-Funktion des MapReduce-Auftrags. Im Folgenden werden einige Beispiele für MapReduce-Aufträge unter Hadoop, gefolgt von kurzen Erläuterungen, gezeigt.

Quellcode (Zählen der Worthäufigkeit in einem Dokumentsatz)

```
14  public class WordCount {
15  /**
16   * The map class of WordCount.
17   */
18  public static class TokenCounterMapper
19      extends Mapper<Object, Text, Text, IntWritable> {
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23      public void map(Object key, Text value, Context context)
24          throws IOException, InterruptedException {
25          StringTokenizer itr = new StringTokenizer(value.toString());
26          while (itr.hasMoreTokens()) {
27              word.set(itr.nextToken());
28              context.write(word, one);
29          }
30      }
31  }
32  /**
33   * The reducer class of WordCount
34   */
35  public static class TokenCounterReducer
36      extends Reducer<Text, IntWritable, Text, IntWritable> {
37      public void reduce(Text key, Iterable<IntWritable> values, Context context)
38          throws IOException, InterruptedException {
39          int sum = 0;
40          for (IntWritable value : values) {
41              sum += value.get();
42          }
43          context.write(key, new IntWritable(sum));
44      }
45  }
46  /**
47   * The main entry point.
48   */
49  public static void main(String[] args) throws Exception {
50      Configuration conf = new Configuration();
51      String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
52      Job job = new Job(conf, "Example Hadoop 0.20.1 WordCount");
53      job.setJarByClass(WordCount.class);
54      job.setMapperClass(TokenCounterMapper.class);
55      job.setReducerClass(TokenCounterReducer.class);
56      job.setOutputKeyClass(Text.class);
57      job.setOutputValueClass(IntWritable.class);
58      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
59      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
60      System.exit(job.waitForCompletion(true) ? 0 : 1);
61  }
62  }
```

Das wohl bekannteste Beispiel für eine MapReduce-Funktion lässt sich unter *Apache Hadoop* leicht und ohne Umwege realisieren. Auffällig sind an diesem Beispiel die *Hadoop*-eigenen Datentypen *Text* und *IntWritable*. Letzteres ist ein Datentyp für Integers, welcher ein eigenes Serialisierungsprotokoll beinhaltet. *Text* ist ein *Hadoop*-eigener Datentyp, welcher Serialisierungsfunktionen sowie die Möglichkeit bringt, einen String auf Byteebene zu durchlaufen.

Die Stelle der Übergabe des Auftrags an den Cluster ist auf den ersten Blick nicht ersichtlich. Dies liegt an der etwas irreführenden Bezeichnung der Methode *waitForCompletion* der Klasse *Job*. Diese ist eigentlich eine Abwandlung der Methode *submit*, welche erst zum Abschluss des Auftrags Daten zurück gibt.

4.3.3 Ausführen von MapReduce-Aufträgen

Hadoop enthält eine Java-Klasse, um MapReduce-Aufträge zu erstellen. Auf diese wurde in Abschnitt 4.3.2 eingegangen. Darüber hinaus ist es unter Hadoop über die Erweiterung *Streaming* möglich, beliebige ausführbare Dateien oder Skripte als Mapper/Reducer zu verwenden, sofern diese (Schlüssel, Wert)-Paare über STDIN/STDOUT verarbeiten. MapReduce-Aufträge lassen sich zum Zeitpunkt der Fachstudie unter *Apache Hadoop* nur über die Kommandozeile ausführen. Theoretisch gibt es für Java-Jobs eine IDE-Integration, jedoch ist diese entweder nicht funktionsfähig, wie das *Hadoop* Eclipse-Plugin, oder gehört nicht zum *Hadoop*-Projekt (Karmasphere). Das *Hadoop* Eclipse-Plugin bietet ein Zugriff auf einen oder mehrere HDFS-Cluster über den Eclipse Explorer, sowie die Möglichkeit, MapReduce-Aufträge direkt aus Eclipse zu kompilieren, auszuführen und zu debuggen. Darüber hinaus wird die Ausgabe des Jobservers direkt in der Eclipse-Konsole ausgegeben.
Die Konfigurationsmöglichkeiten scheinen sehr vielseitig zu sein, da sie jedoch weder dokumentiert noch funktionsfähig sind, lässt sich darüber hinaus keine sinnvolle Aussage treffen.
Unter Java wird die Kommunikation mit dem *Job-Queue* über die Klasse *Job* abgewickelt. Sofern diese mit Hostname oder IP des *JobTrackers* des Clusters versorgt wird, lässt sich der Auftrag von einem beliebigen Rechner aus ausführen. Da *Streaming* ein Teil von *Hadoop* ist, muss *Hadoop* auf derjenigen Plattform installiert sein, von dem aus der Auftrag gestartet werden soll. *Streaming* übernimmt hierbei die Kommunikation mit dem *Jobtracker*, so dass diese in den *Map*- und *Reduce*programmen nicht berücksichtigt werden muss. Die Weiterverwendung der Ergebnisse in weiteren Aufträgen ist unter *Hadoop* problemlos möglich, sofern die Typen der (Schlüssel, Wert)-Paare des Reducers des ersten Aufträgen und des Mappers des zweiten Auftrags kompatibel sind. Da ein *Hadoop*-Cluster über SSH kommuniziert, ist das Hostbetriebssystem eines Knotens weitgehend irrelevant. Vom Einsatz in der Produktion wird mit *Microsoft Windows*-Produkten jedoch abgeraten.

4.3.4 Skalierung

Das *Hinzufügen von Knoten* in *Apache Hadoop* ist simpel, jedoch nur über die Konsole möglich. Nötige Konfigurationsänderungen sind, den SSH-Key des Masterknotens auf den neuen Knoten zu kopieren und Hostname/IP des Masterknotens in die Konfigurationsdatei des neuen Knotens zu schreiben. Wird danach der neue Knoten gestartet, fügt sich dieser automatisch in den Cluster ein.
Soll der Knoten nach einem Neustart des Clusters erhalten bleiben, ist es zudem notwendig, dessen Hostname/IP in die Slaveliste des Masterknotens einzutragen. Der einzige zusätzliche Aufwand für das *Hinzufügen eines Knotens zur Laufzeit* besteht aus dem Ausführen eines Refreshbefehls auf dem Masterknoten. *Ein Knoten kann* aus einem angehaltenen Cluster *entfernt werden*, indem sein Hostname/IP aus der Konfigurationsdatei des Masterknotens entfernt wird. Hadoop besitzt Konfigurationsdateien, um Knoten aus *HDFS* oder MapReduce auszuschließen. Werden Knotenadressen diesen Dateien hinzugefügt, werden diese nach einem Refreshbefehl an den Cluster automatisch abgeschaltet. Der zum Zeitpunkt der Fachstudie größte bekannte Cluster (Facebook) umfasst ca. 2000 Knoten. Angaben zu einer eventuellen Maximalgröße werden seitens des Herstellers nicht gemacht.

4.3.5 Lastbalancierung

Apache Hadoop besitzt in der getesteten Version keinen Mechanismus zur automatischen *Lastbalancierung*. Es ist jedoch möglich, eine manuelle *Lastbalancierung* über die Anzahl der Reducer-Instanzen zu betreiben. Wird diese höher als die Anzahl der Knoten gesetzt, bearbeiten die schnelleren Knoten mehr Reducer-Tasks, so das diese so mehr Arbeitsleistung erbringen können.

4.3.6 Dateisystem

Das Einlesen so wie der Export von Daten aus dem *systemeigenen Dateisystem* ist ohne Umwege mit einem einzelnen Konsolenbefehl möglich. Zusätzlich zum eigenen *verteilten Dateisystem HDFS* unterstützt *Hadoop* Amazon S3 und Cloudstore. Darüber hinaus ist *Hadoop* dazu in der Lage, über FTP oder HTTP externen Speicher zu nutzen, letzteres jedoch nur mit Lesezugriff. *HDFS* kann nicht direkt in das Hostbetriebssystem eingehängt werden. Zugriff ist über die native Java-API, die Konsole oder das WebUI der *NameNode* möglich. Darüber hinaus existiert eine Thrift-API, um *HDFS*-Clients in einer Vielzahl von Sprachen zu implementieren. *HDFS* repliziert Datensätze über eine frei definierbare Anzahl von Knoten (standardmäßig 3). So fern mindestens ein Replikat existiert, wird der Datensatz vom Cluster wieder auf die ursprüngliche Menge repliziert. Ein Sonderfall ist der Ausfall der *NameNode*, des Masterknotens eines *HDFS*-Clusters. In diesem Fall ist der *HDFS*-Cluster nicht mehr lauffähig. Die *NameNode* stellt also einen *Single Point of Failure* dar.

4.3.7 Fehlertoleranz

Beim Verhalten bei einem Knotenausfall muss zwischen zwei verschiedenen Szenarien unterschieden werden: Dem Ausfall eines *TaskTrackers* und dem eines *JobTrackers*.
Der TaskTracker sendet zur Laufzeit einen *Heartbeat*, ein Signal mit dessen Hilfe die Synchronität des Systems überprüft werden kann, an den *JobTracker*. Bleibt dieser aus, so werden eventuell bereits zugeteilte *Tasks* einem neuen Knoten zugeteilt. Die Funktionaliät des Clusters wird nicht beeinträchtigt. Der *JobTracker* stellt einen *Single Point of Failure* dar. Fällt er aus, verliert der Cluster seine Funktionalität. Bereits geleistete Arbeit geht nicht komplett verloren, da der *JobTracker* über ein Checkpointing-System verfügt. Wird die Verbindung zwischen *Job*- und *TaskTracker* unterbrochen, wird der *Task* einem neuen *TaskTracker* zugewiesen.

4.3.8 Monitoring

Apache Hadoop besitzt drei Ansätze zur Überwachung von Clustern: *WebUI*, *Vaidya* und *Metrics*.
Die Daemons *JobTracker*, *TaskTracker* und *NameNode* verfügen über eigene Weboberflächen, welche den aktuellen Status der Daemons tabellarisch wiedergeben. Graphen oder interaktive Elemente sind, abgesehen von der Menüführung, nicht vorhanden. Die WebUIs dienen ausschließlich dem Monitoring, eine Änderung der Konfiguration ist durch diese nicht möglich. *Vaidya* ist ein regelbasiertes Diagnosewerkzeug für MapReduce-Aufträge. Es muss nicht manuell nachkompiliert oder konfiguriert werden und ist standardmäßig nicht Teil von Hadoop, ist jedoch in diversen *Hadoop*-Distributionen enthalten. *Vaidya* nutzt eine Kombination aus XML-Testdefinitionen und vorkompilierten Java-Testklassen, um die Logs/Konfigurationsdateien von MapReduce-Aufträge auf vordefinierte Fehlerszenarien zu überprüfen und gibt den Testbericht als XML-Datei aus. Zur Veranschaulichung der Funktionsweise wird im Folgenden ein Test beschrieben, welcher den Prozentsatz an mehrfach ausgeführten *Map*-Auftrags errechnet, und bei einem Wert von über 40% positiv ist.

Die Vaidya-Testbeschreibungsdatei

```
<PostExPerformanceDiagnosisTests>
 <DiagnosticTest>
        <Title>Impact of Map tasks ReExecution</Title>
        <ClassName>
                org.apache.hadoop.vaidya.postexdiagnosis.tests.MapsReExecutionImpact
        </ClassName>
        <Description>
                This test rule checks percentage of map task re-execution impacting the job
                performance
        </Description>
        <Importance>High</Importance>
        <SuccessThreshold>0.40</SuccessThreshold>
        <Prescription>default advice</Prescription>
 </DiagnosticTest>
</PostExPerformanceDiagnosisTests>
```

Das Element "Importance" kann die Werte *Low, Medium* und *High* annehmen. Dieser Wert wird in der Ausgabe genutzt um in der Ausgabe die Schwere des Fehlers zu kalkulieren. Das Element "Prescription" enthält Ratschläge zur Fehlerbehebung, welche in die Ausgabe-XML geschrieben werden, falls der Test positiv ist.

Die Java-Testklasse

```
public class MapsReExecutionImpact extends DiagnosticTest {

  private double _impact;
  private JobStatistics _job;
  private long _percentMapsReExecuted;

  /**
   *
   */
  public MapsReExecutionImpact() {
  }

  /*
   * Evaluate the test
   */
  @Override
  public double evaluate(JobStatistics job) {
    this._job = job;

    /*
     * Calculate and return the impact
     */
    this._impact = ((job.getLongValue(JobKeys.LAUNCHED_MAPS) -
 job.getLongValue(JobKeys.TOTAL_MAPS))/job.getLongValue(JobKeys.TOTAL_MAPS));
    this._percentMapsReExecuted = Math.round(this._impact * 100);
    return this._impact;
  }

  @Override
  public String getPrescription() {
    return
    "* Need careful evaluation of why maps are re-executed. \n" +
      "  * It could be due to some set of unstable cluster nodes.\n" +
      "  * It could be due application specific failures.";
  }

  @Override
  public String getReferenceDetails() {
    String ref = "* Total Map Tasks: "+this._job.getLongValue(JobKeys.TOTAL_MAPS)+"\n"+
                 "* Launched Map Tasks: "+this._job.getLongValue(JobKeys.LAUNCHED_MAPS)+"\n"+
                 "* Percent Maps ReExecuted: "+this._percentMapsReExecuted+"\n"+
                 "* Impact: "+truncate(this._impact);
    return ref;
  }
}
```

Von Interesse ist hier besonders die Funktion *evaluate*, welche den eigentlichen Testfall darstellt. Der Ausgabewert ist der *impact level*, welcher mit dem in der Testbeschreibungsdatei definierten *success threshhold* verglichen wird. Ist ersterer über letzterem, so wird der Test als positiv ausgegeben. Die einzelnen *Hadoop*-Daemons (*NameNode, SecondaryNameNode, DataNode, JobTracker*, und *TaskTracker*) verfügen über abrufbare Laufzeitmetriken. Diese können entweder als Logfile ausgegeben, oder direkt via JMX (MBeans oder RPC) vom Daemon abgerufen werden (zum Beispiel mit dem JDK beiliegenden Jconsole). *Hadoop* organisiert seine Metriken in so genannte *Contexts*, welche die Metriken gliedern. Diese können auch von Benutzer erzeugt werden. Dies machen sich Hersteller von Drittsoftware (Ganglia, Nagios) zunutze, um aussagekräftigere Metriken zu erstellen.

4.3.9 Fazit

Hadoop ist eine überaus mächtige MapReduce-Implementierung. Die schiere Anzahl an Möglichkeiten jeglicher Kategorie, sei es das dedizierte Testframework Vaidya, Zugriffsmöglichkeiten auf verschiedene verteilte Dateisysteme oder Java-Klassen für jeden erdenklichen MapReduce Use-Case ist beträchtlich. Allerdings merkt man *Hadoop* seine Open-Source/Linux Herkunft deutlich an. Die Konfigurations- und Monitoringmöglichkeiten sind enorm, jedoch nur über die manuelle Manipulation von Textdateien zugänglich. Auch der Zustand der *Dokumentation* und der Zusatzmodule ist mit hoher Wahrscheinlichkeit darauf zurückzuführen, dass für diese nicht genügend Entwicklerstunden zur Verfügung stehen. *Hadoop* lief über den gesamten Testzeitraum stabil, nachdem es gegen die Testumgebung kompiliert wurde, Abstürze kamen nicht vor.
Für den produktiven Einsatz ist *Hadoop* durchaus empfehlenswert. Die vielfältigen Konfigurationsmöglichkeiten, so wie die Flexibilität bezüglich der verwendeten Programmiersprachen machen diese Implementierung zu einem ausgezeichneten Kandidaten. Durch die etwas umständliche Konfiguration und Wartung, sowie die große Zahl an nachinstallierbarer Drittsoftware bietet es sich jedoch an, *Hadoop* nicht in der Ursprungsversion, sondern im Rahmen einer der vielen "fertigen" Distributionen (wie zum Beispiel dem in dieser Fachstudie behandelten *MapR*) zu verwenden.

4.4 MapR

Als nächstes möchten wir die *Hadoop*-Distribution *MapR* näher betrachten, welche sich wie im *Abschnitt 3.5* und *3.6* dargelegt gegenüber seinem Hauptkonkurrenten *Cloudera* behaupten konnte.

4.4.1 Installation, Inbetriebnahme und Dokumentation

Bevor man *MapR* installieren kann, muss man sich entweder für die Opensource *MapR* M3 Edition oder die kommerzielle M5 Edition entscheiden. Als nächstes hat man die Wahl zwischen einer vorinstallierten und konfigurierten Viritual-Machine-Version, die man nur per VMware starten und nutzen kann. Oder man installiert *MapR* manuell auf eine 64-Bit Linuxdistribution. Clients zur Überwachung und Bedienung von Clustern, die mit *MapR* erstellt wurden, stehen zusätzlich für OSX und Windows 32/64 Bit zur Auswahl. Die *Installation* und *Konfiguration* der M3-Virtual-Machine-Version beschränkt sich auf die Installation von *VMware*, die Vergabe der Administratorrechte an die virtuelle Maschine und das Starten der bereits vorkonfigurierten *MapR* M3-Version. Diese basiert auf Ubuntu 10.04 LTS 64 Bit und enthält einen Hadoop-Cluster mit einem Knoten und mehreren Partitionen, die grafische Bedienoberfläche und der Einbindung des

Clusters im Dateisystem. Man kann *MapR* entweder auf einem einzelnen oder auf mehreren Rechnern installieren und sie zu einem Cluster vereinen. Beide Installationsvarianten setzen das *Java JDK* voraus und werden über das MapR eigene Onlinerepository installiert. Alle *Hadoop*-Komponenten existieren hierfür als MapR-Versionen, wie z.B. dem *MapR-Fileserver*, *MapR-Jobtracker*, *-Tasktracker* usw. Und für jede existiert eine Installations- und Bedienungsanleitung in der *Dokumentation*. Aufgrund des Distributionscharakters von *MapR* erwies sich die *Installation* als reibungsfrei. Die *Inbetriebnahme* der Virtual-Machine-Version läuft reibungsfrei ab, da alle relevanten Einstellungen bereits vorkonfiguriert sind. Bei einer eigenen Installation müssen die vorhandenen Festplatten noch eingebunden, die *Zookeeper*-Komponente und ein Prozess namens *Warden* gestartet werden. Letzterer startet wiederum ganz automatisch alle anderen *MapR*/*Hadoop*-Komponenten und sorgt im Falle eines Ausfalls dafür, dass diese erneut gestartet werden und das gesamte System funktionsbereit ist.
Die Online-*Dokumentation* beschreibt mit Text und Bildern sowohl die *Installation* der verschiedenen MapR-Versionen auf verschiedenen Betriebssystemen, als auch deren Konfiguration in klaren Schritten. Sie gibt dazu erste Beispiele der Handhabung des *MapR Control Systems*, der Erstellung von *Volumes*, dem Import und Export von Daten auf die *Volumes* im Cluster über *MapR-NFS* und deren Bearbeitung mittels MapReduce-Aufträgen. Eine Einführung zu Snapshots, Datenspiegelungen, Zeitplänen als auch zu *HBase*, *Hive* und *Pig* ist ebenso enthalten. Tiefer greifende Anleitungen und Informationen erhält man in den Dokumentationsbereichen Installation, Administration, Entwicklung, Migration und Referenz. Insgesamt war die *Dokumentation* sehr hilfreich und ermöglichte eine reibungslose Einrichtung und Bedienung von *MapR*.

4.4.2 Erstellung von MapReduce-Funktionen

Die Erstellung von MapReduce-Funktionen geschieht auf die gleiche Weise wie bei *Hadoop*. Eine ausführliche Erläuterung dazu ist im Abschnitt 4.3.2 zu finden. Die einzelnen Map- und Reduce-Klassen werden mit der entsprechenden Funktionsklasse, z.B. *WordCount* als eine ausführbare JAR-Datei erstellt und über die Konsole ausgeführt. Anzugebende Parameter in der Befehlszeile sind: *`hadoop jar "Pfad-Jar-Datei" "Funktionsname=Klassenname" "Pfad-Eingabedateien" "Pfad-Ausgabeordner"`*
Mit dem Befehl *hadoop* beginnt jede MapReduce-Funktion und mit *jar "Pfad-Jar-Datei"* wird der Typ und der Pfad zur ausführbaren Datei, welche die MapReduce-Funktion enthält, angegeben. Darauf folgt die konkrete Wahl der Funktion, zum Beispiel mit dem *"Klassennamen" Wordcount* und als letztes wird der Pfad zu den Eingabedateien und der Ausgabepfad für die bearbeiteten Ergebnisse übergeben. Für einen erweiterten Bedienkomfort ist es mittels Apache 2 & Ajaxterm möglich die Befehlskonsole direkt in die Hauptbedienoberfläche namens *MapR Control System* zu integrieren.

4.4.3 Ausführen von MapReduce-Aufträgen

MapR unterstützt von Haus aus sowohl Java als auch beliebige andere Sprachen für die Ausführung von MapReduce-Aufträgen, falls die Erweiterung *Streaming* installiert ist. MapReduce-Funktionen für MapR entsprechen denen von Hadoop und entsprechend müssen diese als vorkompilierte und ausführbare JAR-Datei vorliegen. Die Entwicklung von entsprechenden Klassen und Funktionen werden von *MapR* nicht zusätzlich unterstützt und müssen auf eigene Art und Weise in einer IDE nach Wahl programmiert werden.
Die Übergabe der Aufträge an die *Job-Queue* entspricht der von *Hadoop*. Wichtig ist v. a. dass sich

alle Daten, die analysiert oder bearbeitet werden sollen, im *MapR NFS* befinden. Da alle Auftrags-Ausgaben auch direkt im Dateisystem sichtbar sind, da sie je nach MapReduce-Auftrag etwa in Form von Textdateien vorliegen, können diese auf die selbe Weise als Eingabedaten für neue MapReduce-Aufträge weiterverwendet werden. Alternativ können die Ausgabedateien per Dateibrowser betrachtet und mit vorhandenen Bearbeitungsprogrammen editiert werden.
Die Systemlast, sowie auch der Speicherbedarf, wird von *MapR-Hadoop* unabhängig der Hardwarebestückung einzelner Rechner so optimal wie möglich verteilt und spielt daher für die Ausführung von MapReduce-Aufträgen und der Speicherung von Daten im *MapR NFS* keine Rolle. Zusätzlich dazu bietet *MapR* Clients für Linux, Windows und Mac OS an und gewährleistet damit plattformübergreifende Bedienmöglichkeiten eines mit MapR erstellten *Hadoop*-Clusters.

4.4.4 Skalierung

Das *Hinzufügen eines Knotens* in ein *MapR*-Cluster erfordert die Installation von *MapR* mit mindestens der *Fileserver*- und der *Tasktracker*-Komponente auf dem Knoten und der Konfiguration der Festplatten in der Datei *disks.txt* und der Ports in der Datei *configure.sh*. Anschließend müssen die Festplatten im Knoten formatiert und der *MapR*-Prozess Warden gestartet werden. Der Knoten wird automatisch erkannt und dem Cluster hinzugefügt. Zur Laufzeit ist dies auch möglich, sofern ein Refreshbefehl im Masterknoten, der alle *MapR*-Komponenten im Betrieb hat, eingegeben wird. Bereits laufende MapReduce-Aufträge werden von dieser Maßnahme nicht beeinflusst und profitieren erst beim nächsten Auftrag von der neuen Kapazität.
Knoten können aus einem angehaltenen Cluster durch das Streichen der IP-Adresse in der Konfigurationsdatei der Knoten mit der ZooKeeper-Komponente entfernt werden. *Das Entfernen eines Knotens zur Laufzeit* kann im Gegensatz dazu ganz leicht über die grafische Bedienoberfläche in der Knoten-Übersicht durchgeführt werden, indem alle *MapR*-Dienste gestoppt werden, was mit dem Testsystem (siehe *Abschnitt 4.2*) bis zu einigen Minuten dauerte. Anschließend kann der aktiv gewordene "Remove"-Button betätigt werden. Alternativ kann ein Knoten auch per Konsolenbefehl vom Cluster entfernt werden.
Laut Aussagen auf der Webseite von *MapR* skaliert die Speicherkapazität und die erlaubte Menge von einzelnen Dateien linear mit jedem weiteren Knoten [MRS]. Beides sei prinzipiell unbegrenzt. In einer Tabelle gibt die Seite weiter den Hinweis, dass normale *Hadoop*-Cluster auf 3000 Knoten begrenzt seien und mit MapR mehr als 10.000 Knoten möglich wären [MRS]. Leider können wir im Rahmen dieser Fachstudie keine der Aussagen überprüfen.

4.4.5 Lastbalancierung

Auch wenn *Hadoop* von sich aus keine automatische Lastbalancierung unterstützt, ermöglicht MapR durch eigene Algorithmen das Erkennen von sehr kleinen MapReduce-Aufträgen und kann trotz voller MapReduce-Slots automatisch einige Slots freischalten und den Auftrag parallel, aber mit hoher Priorität durchführen. In einem klassischen *Hadoop*-Cluster würde der neue Auftrag einfach an die *Job-Queue* angehängt werden. Diese Möglichkeit wird *MapR ExpressLane* genannt und kann aktiviert und konfiguriert werden.

4.4.6 Dateisystem

Der Import und Export von Daten aus und in das systemeigene Dateisystem geschieht entweder über die Konsole oder per Drag-and-Drop über den Betriebssystem-eigenen Dateimanager. *Hadoop*-Cluster, die auf *HDFS* basieren, sowie die darauf erstellten *MapR Volumes* können mittels *MapR NFS (Network File System)* bequem im Dateibrowser des Betriebssystems eingehängt

werden. Sie verhalten sich dann wie Ordner, auf die man schreibend und lesend zugreifen kann um z.B. Testdateien auf dem Cluster abzulegen und diese mittels MapReduce-Aufträgen zu bearbeiten.

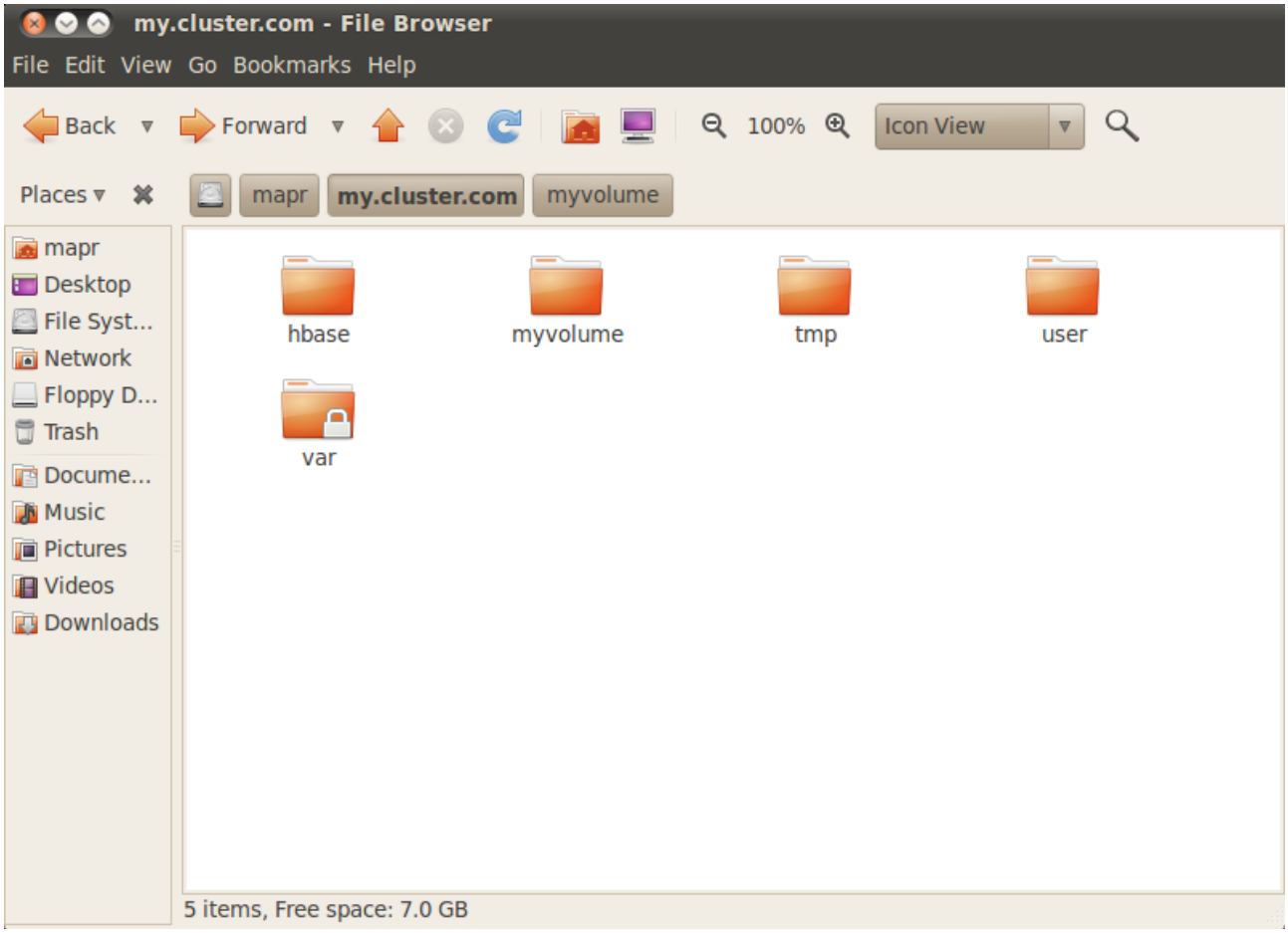

Abbildung 4.4.6.1: MapR NFS vereint Cluster und Cluster-Volumes zu einer Ordnerstruktur

In der oberen Abbildung wurde der *MapR*-Cluster "my.cluster.com" und eine Partition namens "myvolume" eingehängt und beide zeigen sich nun in einer Ordnerstruktur. Seit neuestem (Stand der Fachstudie) bietet *MapR* auch die Unterstützung der Amazon- und Google-Cloud an, letztere befindet sich aber noch in der Betaphase [MRA][MRG]. *MapR* bietet in der freien M3-Version zudem automatische Replikationen der *Volumes* und in der M5-Version *Snapshots* und *Spiegelungen* an, um die Daten bei Knotenausfällen wiederherstellen zu können. *Snapshots* sind dabei Kopien von *Volumes*, die zu einer bestimmten Zeit erstellt wurden und automatisch z.B. am Ende eines Arbeitstages alle Daten sichern. *Spiegelungen* der *Volumes* dagegen sind Echtzeitkopien, welche standardmäßig dreifach erstellt werden und bei einem Knotenausfall Verzögerungen und Datenverlust bei MapReduce-Aufträgen verhindern. Dieser Wert kann für jedes *Volume* einzeln oder für alle über die Web-GUI verändert werden, siehe rote Markierung in der folgenden *Abbildung 4.4.6.2*.

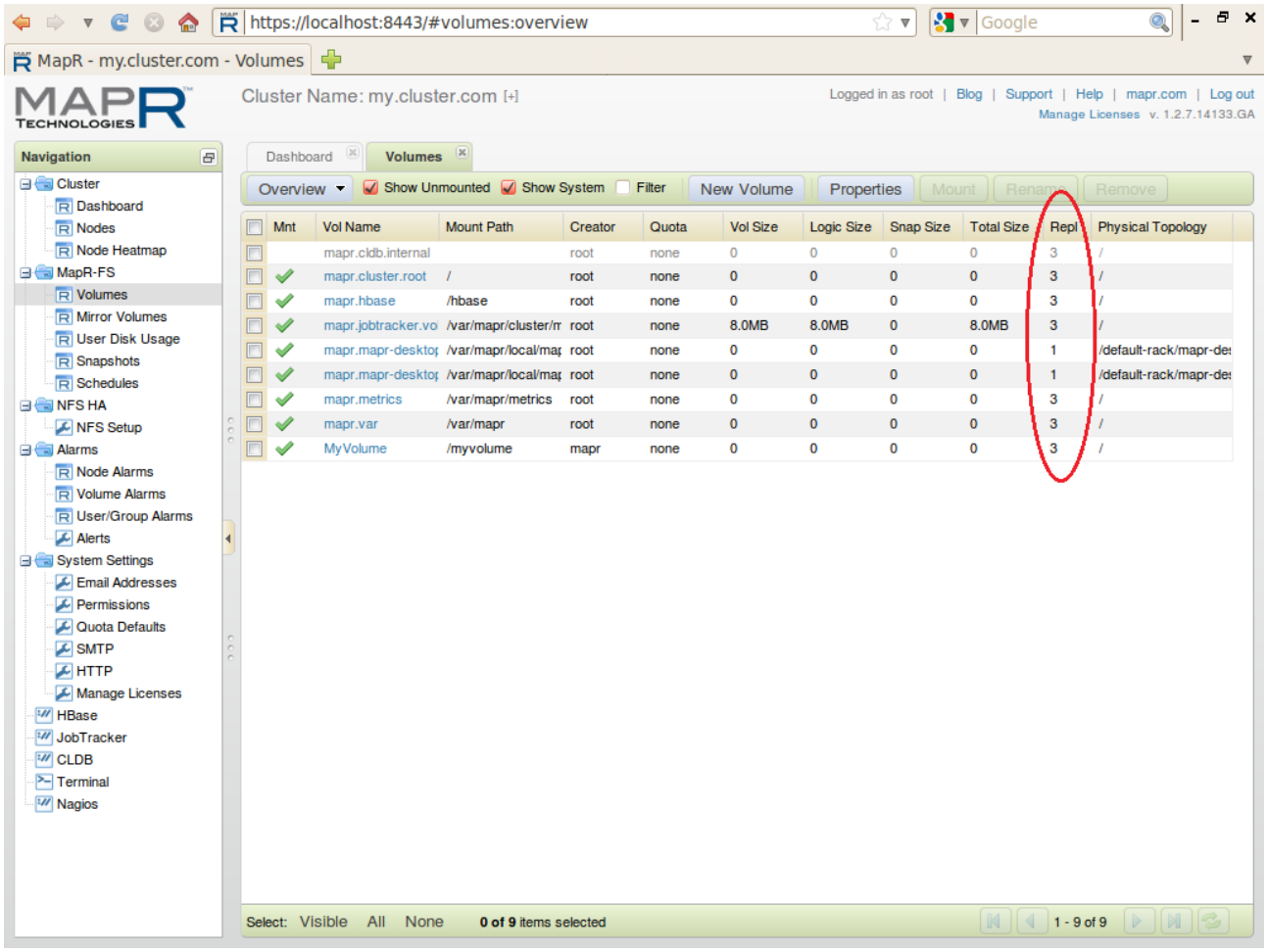

Abbildung 4.4.6.2: Die Replikationszahl der einzelnen Volumes im Cluster

4.4.7 Fehlertoleranz

Das Verhalten bei, auch kurzzeitigem, Ausfall eines Knotens entspricht dem von *Hadoop* und wird im *Abschnitt 4.3.7* behandelt. Darüber hinaus wird jeder Ausfall visuell in der Web-GUI, auf die im nächsten Abschnitt ausführlicher eingegangen wird, dargestellt, protokolliert und der Benutzer wird zusätzlich durch ein Alarmsignal informiert.

4.4.8 Monitoring

MapR Control System nennt sich die grafische Bedienoberfläche, welche die Überwachung und Steuerung von *Hadoop*-Clustern innerhalb eines beliebigen Web-Browsers ermöglicht. Die Adresse des Clusters wird hierfür in die Adresszeile eingegeben und sofort erhält man detaillierte Auskunft über Knoten im Cluster, Volumes, Spiegelungen, Speicherplatzverbrauch, Arbeitslast der Knoten, wartende/laufende und beendete MapReduce-Aufträge und vieles mehr. Nach dem Start der Web-GUI werden zunächst Nutzername und Passwort abgefragt und schließlich landet man mit entsprechenden Rechten in der *Monitoring*-Übersicht, hier als *Dashboard* bezeichnet.

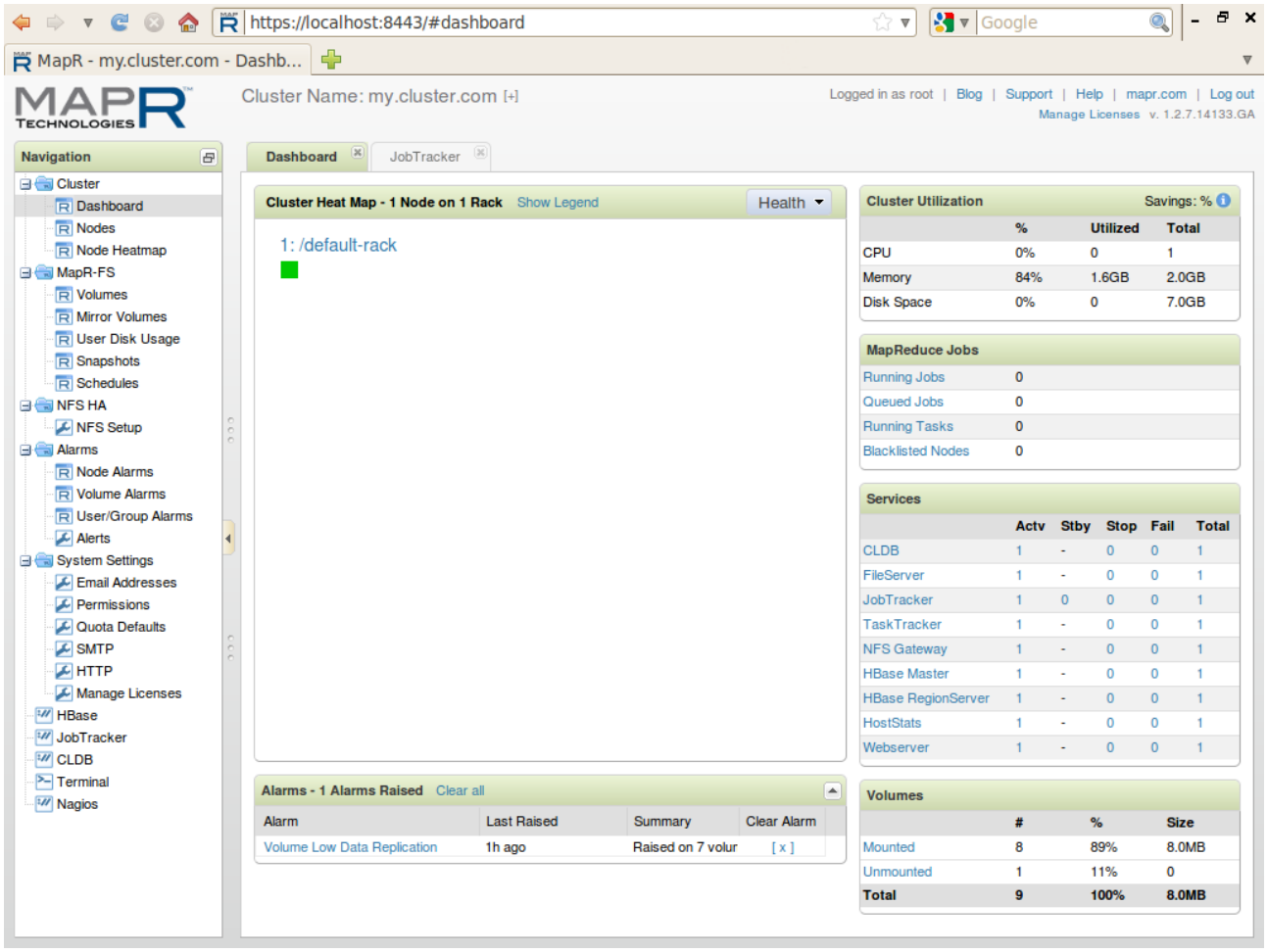

Abbildung 4.4.8.1: Startseite von MapR Control System

Im Navigationsfeld auf der linken Seite der oberen Abbildung können alle installierten Funktionen direkt aufgerufen werden. Diese werden dann als Tabs in der Mitte der Oberfläche dargestellt. Rechts oben gibt es eine kleine Übersicht zur Clusterauslastung, darunter sieht man laufende und wartende MapReduce-Aufträge. Unter dem Punkt *Services* werden alle installierten *MapR*/*Hadoop*-Komponenten und deren Status gelistet. Detailliertere Informationen über die einzelnen Knoten im Cluster erhält man über die Desktop-Ansicht, die in *Abbildung 4.4.8.2* gezeigt wird.

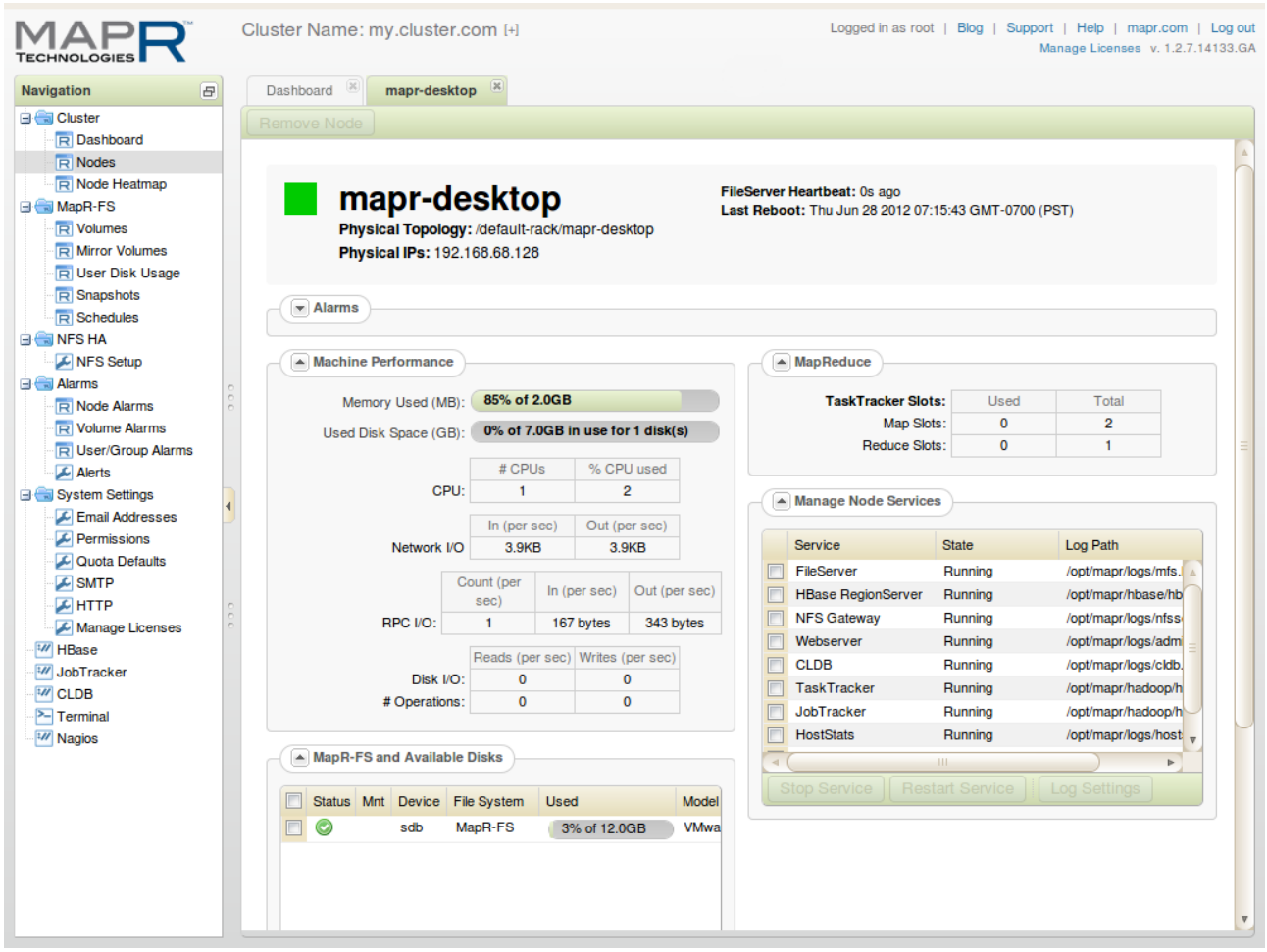

Abbildung 4.4.8.2: Detaillierte Knotenansicht

Neben dem vorhandenen Restspeicherplatz werden hier auch die Anzahl der Prozessorkerne, Map - und Reduce-Slots sowie der Datenverkehr überwacht. Falls irgendwelche Dienstkomponenten nicht funktionieren, wird dies im Feld rechts unten dargestellt. Das *MapR Control System* ist fester Bestandteil von *MapR*, wird automatisch mitinstalliert und integriert alle Komponenten ganz von selbst. Die Bedienung ist intuitiv und auch über externe MacOS X oder Windows-Clients mit Browser möglich. Insgesamt stellt es eine Bereicherung des gesamten *Hadoop*-Projekts dar und vereinfacht dessen Anwendung nach unseren Tests sehr beachtlich. Neben seiner Funktion *Hadoop*-Cluster zu vereinen und im Dateibrowser als Speicherplatz zur Verfügung zu stellen, ist es mittels *MapR NFS* auch möglich Vorgänge von MapReduce-Aufträgen und anderen *Hadoop*-Erweiterungen zu überwachen und zurückzuverfolgen.
In der nächsten *Abbildung 4.4.8.3* wurde die Ausgabe eines MapReduce-Auftrags, dessen Erfolgsbestätigung und die dazugehörigen Log-Dateien in einem gewählten Ordner abgelegt. Falls die Web-GUI mal nicht erreichbar sein sollte, können mittels Konsole oder Dateimanager weiterhin Ergebnisse aufgerufen und Fehler zurückverfolgt werden.

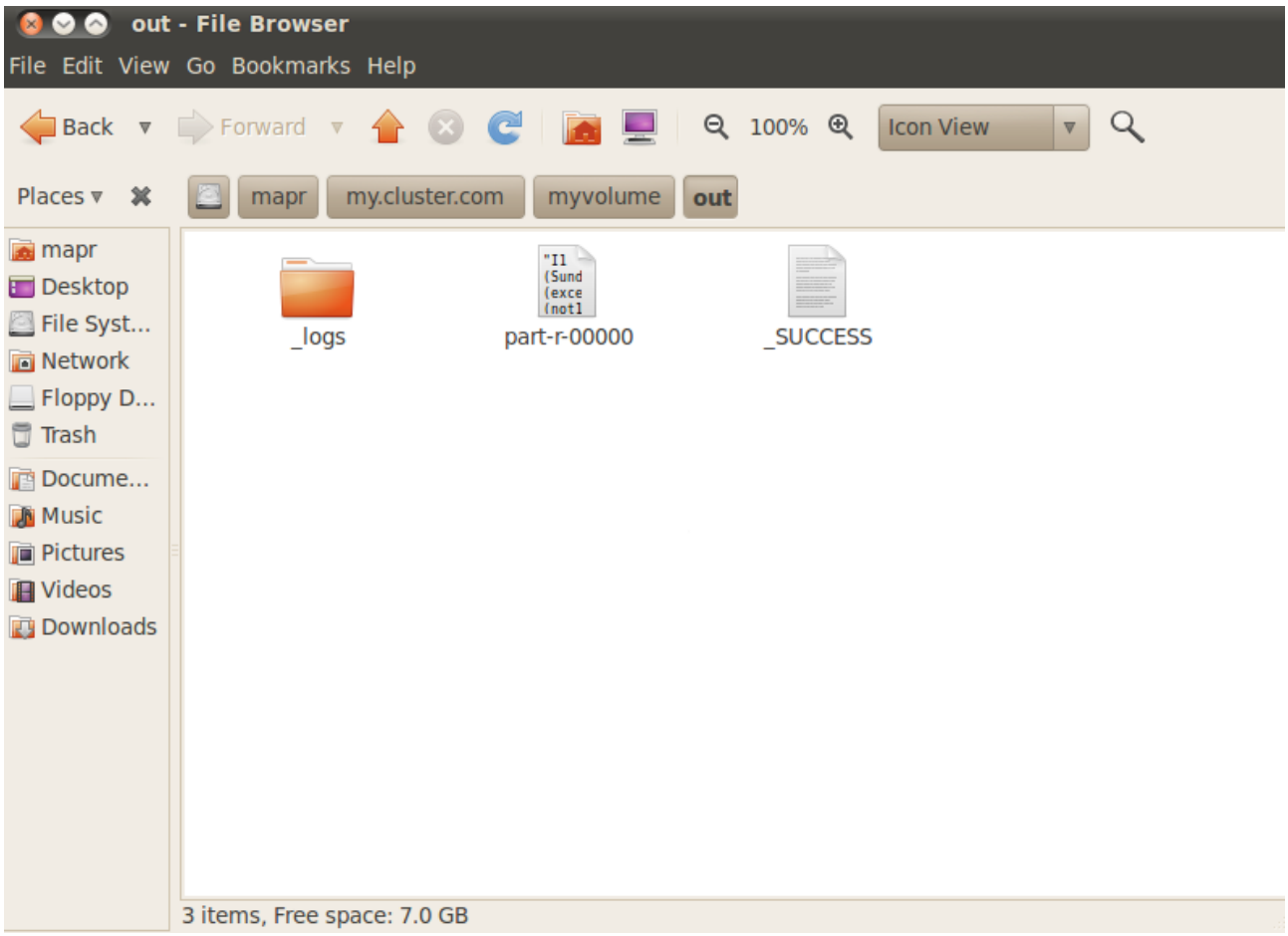

Abbildung 4.4.8.3: Auftragsdetails und Prozessfeedbacks mittels MapR NFS

4.4.9 Sonstiges

Es gibt sehr viele Erweiterungen von Apache, die den Funktionsumfang von Hadoop deutlich erweitern. Normalerweise ist sowohl deren Installation, Konfiguration als auch deren Benutzung relativ umständlich gelöst. Da *MapR* mit einer leichten Integration dieser Erweiterungen wirbt, und sogar Varianten der Software anbietet, in denen diese bereits enthalten sind, möchten wir im Folgenden die bekanntesten drei näher untersuchen.

Apache *HBase*

Die Installation von *HBase* für die Linuxdistributionen Ubuntu, Red Hat und CentOS geschieht über die *MapR*-Repositories. Ein Konsolenbefehl reicht aus um einen oder mehrere Nodes zu einem *HBase* Master oder RegionServer zu erweitern. Mit *HBase* können sehr große Tabellen erstellt werden, welche nicht nur Daten speichern können, sondern u. a. auch für MapReduce-Aufträge zur Verfügung stehen. Dabei werden sogenannte "Spaltenfamilien" definiert, die wiederum aus vielen Spalten bestehen, welche Zeilen mit Einträgen aufnehmen können. Die Erstellung und Befüllung dieser Tabellen geschieht über die *HBase*-Konsole, welche über die Linux-Konsole gestartet wird. Die Befehle sind dabei recht einfach gehalten, z.B. *create '<Tabellenname>', '<Spaltenfamilienname>'.* Mit *list* können alle Tabellen aufgelistet, mit *put* befüllt und mit *scan* '<Tabellenname>' betrachtet werden. Für die gängigsten MapReduce-Aufträge stellt das *HBase*-Projekt eine eigene Java-Klasse zur Verfügung, die mit folgendem Konsolenbefehl den Zugriff auf die Tabellen ermöglicht:

hadoop jar "Pfad-ausführbare-HBase-Jar-Datei" "Funktionsname-Export" "Tabellenname" "Pfad-Ausgabeordner". Bei diesem Beispiel wird eine Tabelle in das *Hadoop*-Dateisystem exportiert und kann mittels *MapR NFS* im Dateibrowser betrachtet werden. Die Überwachung der Cluster, der *HBase* Knoten mit detaillierten Informationen als auch der MapReduce-Aufträge ist dabei über die *MapR* Web-GUI möglich.
Insgesamt ist der Einstieg in die Installation und Nutzung von *HBase* innerhalb der *MapR*-Distribution aufgrund der guten Dokumentation und dem eigenen Repository recht einfach. Zudem wird die Erweiterung nahtlos in die grafische Bedienoberfläche integriert und kann mithilfe von speziellen Terminals, wie z.B. *Ajaxterm* auch damit gesteuert werden.

Apache *Hive*
Die Installation von *Hive* geschieht auf die gleiche einfache Weise wie *HBase* über das *MapR* Repository mittels eines Konsolenbefehls. Durch zwei weitere Befehle werden die Umgebungsvariablen für *Hive* und *Java* angepasst und dann ist es betriebsbereit. *Hive* ermöglicht die Erstellung von Tabellen, deren Befüllung und Abfrage durch eine Syntax namens *HiveQL*, welche derjenigen von SQL sehr stark ähnelt. Die Bedienung geschieht dabei wieder über die Linux-Konsole. Mit *CREATE TABLE <Tabellenname>(<Spaltenname1> <Datentyp1>, <Spaltenname2> ..,..)* können Tabellen erstellt werden, mit INSERT INTO befüllt und mit den bekannten SQL Statements *SELECT, FROM, WHERE* betrachtet werden. Tabellen können auch mit Inhalten aus anderen Text- oder Tabellenkalkulations-Dokumenten im *Hadoop*-Cluster befüllt oder zu solchen exportiert werden. Zu beachten ist dabei nur, dass sich die Daten und Tabellen sich in den selben Cluster-*volumes* befinden. Um auch sehr große Tabellen abfragen zu können, ist es möglich *HBase* und *Hive* zu verbinden um sogenannten *Hive*-HBase-Tables zu erstellen oder vorhandene *HBase*-Tabellen in solche zu konvertieren. Soweit wir beobachten können, werden die Tabellenabfragen nicht als MapReduce-Aufträge gelistet und sind daher auch in der Web-GUI nicht sichtbar. Mit großer Wahrscheinlichkeit ändert sich dieses Verhalten bei der Integration von *HBase* in *Hive*, kann mangels eines Tests aber nicht bestätigt werden.
Insgesamt ist die Installation und Bedienung von *Hive* gut dokumentiert und daher einfach. Es sollte v. a. erfahrene SQL-Nutzer ansprechen. Weniger gut allerdings ist das Fehlen der Sichtbarkeit der Abfragen als MapReduce-Aufträge im *MapR Control System*, dafür können die exportierten Tabellen sofort im entsprechenden Volume-Ordner des Dateisystems begutachtet und weiter verwendet werden.

Apache *Pig*
Die Installation von *Pig* geschieht auf gleiche Weise wie *HBase* und *Hive*, d.h. über das *MapR* Repository mit einem Konsolenbefehl. Mit *Pig* können große Datensets parallel analysiert werden. Dafür wird eine eigene Sprache namens *PigLatin* mitgeliefert, welche unterschiedliche Befehle als einzelne Buchstaben in die Konsole übergibt und letztendlich einen MapReduce-Auftrag startet. Dieser wird auch in der Web-GUI von *MapR* überwacht. Ein kurzer Vergleich zwischen dem *WordCount*-MapReduce-Auftrag von *Hadoop* selbst und mit *Pig* über *Hadoop* ergab auf dem selben Testsystem einen Geschwindigkeitsvorteil von etwa 100% für *Hadoop*. Es konnte nämlich die vorhandenen zwei *Map*-Instanzen des Testclusters ausnutzen, während *Pig* nur eins davon nutzte, wie man in der folgenden *Abbildung 4.4.9.1* des *JobTrackers* in der Web-GUI sehen kann.

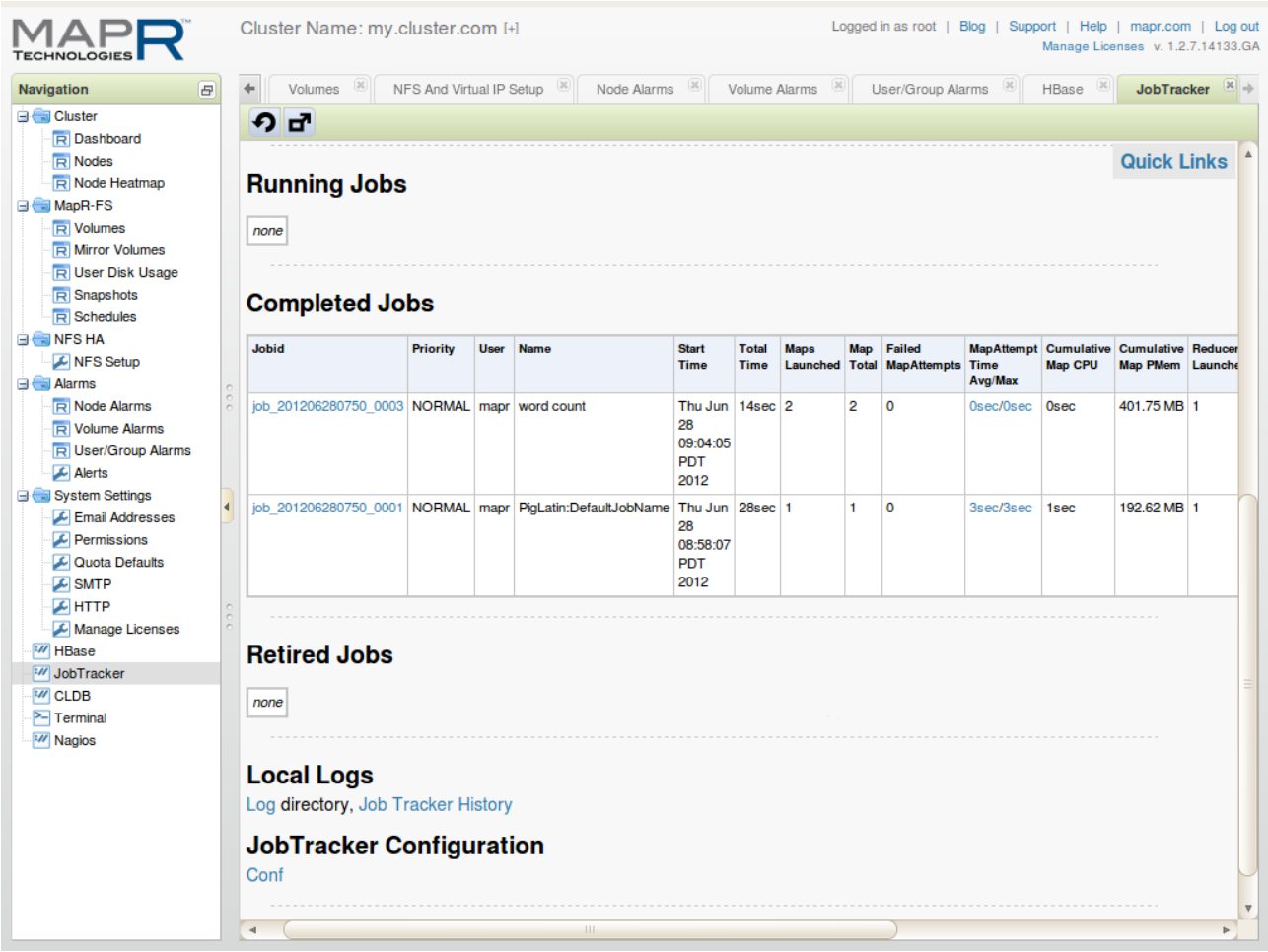

Abbildung 4.4.9.1: Der JobTracker innerhalb von MapR Control System

Die genaue Ursache für dieses Verhalten konnte nicht geklärt werden, vermutlich reserviert sich *Pig* aber einen *Map*-Slot für andere Zwecke und in großen Clustern spielt dieser Faktor keine Rolle mehr. Die Integration von *Pig* in die *MapR*-Umgebung ist wie bei *HBase* gut dokumentiert und erscheint nach unserem Test sehr gelungen.

4.4.10 Fazit

Von allen Implementierungen machte MapR bereits in der ersten Bewertung, im Abschnitt 3 dieser Fachstudie, einen sehr guten Eindruck und erhielt entsprechend hohe Punktzahlen. Das dieser Eindruck nicht täuschte, hat sich mit dieser tiefer gehenden Untersuchung bestätigt. Angefangen von der reibungslosen Installation aller *Hadoop*- und *MapR*-Komponenten über das eigene Repository, dem komfortablen Import beliebiger Daten in den Cluster per Dateibrowser des Betriebssystems, bis hin zur Ausführung des ersten MapReduce-Auftrags, verlief alles reibungslos. Unterstützt wird dies durch eine beispielhaft aufgebaute *Dokumentation*, die jedes Szenario detailliert und mit Bildern und Beispielen beschreibt. Alle Vorteile von *Hadoop*, welche die immense Auswahl der Erweiterungen wie z.B. *Hive*, *Pig* und *HBase* mit einbezieht, werden von *MapR* in einer Distribution vereint und bietet durch die grafische Web-UI nicht nur Möglichkeiten zur Überwachung, sondern ist auch gut konfigurierbar und gebrauchstauglich. Uns gefiel vor allem, dass zusätzliche Erweiterungen einfach aus dem Repository nachinstalliert werden konnten und sich diese automatisch in das Gesamtsystem integrierten.
Der Support schickte während der Testphase regelmäßig Tipps und Tutorials sowie eine Einladung,

weitere Nutzungsbereiche von *MapR* zu erforschen, per E-Mail. Mit großen Partnern wie IBM, Google, Cisco und Amazon und dem professionellen Auftreten über die eigene *MapR*-Webseite, den angebotenen Trainingskursen und dem 24h-Support verspricht MapR seinen Kunden eine stabile Servicebasis, lässt sich diese aber auch bezahlen.
Jedoch ist sind uns auch einige negative Aspekte beim Untersuchen der Implementierung begegnet. Als erstes fällt auf, dass eine erhöhte Datensicherheit durch manuelle Snapshots und ganzen Clusterspiegelungen nur mit der kommerziellen Version zur Verfügung stehen. Gleiches gilt für die Features *NFS HA* und *ExpressLane*, welche aus diesem Grund nicht getestet werden konnten. Auf der anderen Seite sind diese Features eigene Erweiterungen von *MapR* und entstammen nicht der Open-Source-Welt. Insgesamt hat *MapR* als Implementierung überzeugt und sollte bei der Entscheidungsfindung einer MapReduce-Implementierung auf jeden Fall in die engere Auswahl miteinbezogen werden.

4.5 GridGain

Diese flexible Implementierung fällt unter anderem durch die Unterstützung mehrere Programmiersprachen, Betriebssysteme und IDEs auf. Auch das Konzept des In-Memory *DataGrids* machen *GridGain* zu einem interessanten Kandidaten. Hierbei werden Daten konstant im Hauptspeicher der Knoten gehalten, um eine schnelle Verarbeitung zu gewährleisten. Dabei kann die Software mit großen Datenmengen in Echtzeit rechnen und skaliert zudem voll automatisch.

4.5.1 Installation, Inbetriebnahme und Dokumentation

Der Download der von uns untersuchten Version umfasste ca. 100MB plus den Paketen der optionalen Programmiersprachen *Scala* und *Groovy*. Die Installation auf Windows 7 gestaltet sich sehr einfach, hinterher müssen lediglich noch die Umgebungsvariablen *GRIDGAIN_HOME* und J*AVA_HOME* manuell gesetzt werden. Die *Installationsanleitung* erklärt Schritt für Schritt das Einrichten der Entwicklungsumgebung und Ausführen des ersten Beispiels, die Schritte sind dabei verständlich und die erste Anwendung läuft nach kurzer Zeit. Auf der Webseite von *GridGain* findet sich das Online-Buch “Real Time Big Data Processing with GridGain” welches ausführlich auf die Möglichkeiten, die *GridGain* bietet, eingeht sowie auch durch die *Installation*, *Konfiguration* und den Start in die Entwicklung mit *GridGain* führt. Das Buch ist relativ ausführlich und beantwortet viele Fragen die auch im Entwicklungsprozess entstehen können, außerdem sind Erklärungen zu den vielzähligen APIs auch mit internen und externen Links versehen. Jedoch sind einige Kapitel des Buches lückenhaft, so sind beispielsweise einige Unterkapitel des Monitorings noch leer. Weiterhin findet man auf der Webseite einige hilfreiche Tutorial-Videos sowie ein zur Zeit unseres Tests aktiv besuchtes Online-Forum, in dem auf Fragen und Probleme mit der Software eingegangen wird.

4.5.2 Erstellung von MapReduce-Funktionen in Java

In einer beliebigen Java-IDE wird ein neues Projekt erstellt und die *GridGain* JAR-Datei sowie der "libs" Ordner aus dem *GridGain* Verzeichnis als Bibliothek eingefügt. Das Erstellen einer neuen MapReduce-Funktion geschieht durch das Hinzufügen einer neuen Java-Klasse und dem Import der benötigten *GridGain*-Bibliotheken. Dann wird die gewünschte Funktion in Java implementiert. Bei der Implementierung des *Map*-Schrittes kann auf die *yield*-Methode zurückgegriffen werden:

```
F.yield(Collection, GridClosure);
```

Collection ist die Sammlung der zu mappenden Elemente, *GridClosure* ist die zu ausführende Funktion. F.*yield* nimmt jedes Element aus der *Collection* und übergibt sie an die *GridClosure*

Funktion. Zurückgegeben wird dann eine *Collection* mit den Ergebnissen dieser Aufrufe. Der Reduce-Schritt erfolgt per Zugriff auf das *Grid* per Aufruf der Reduce-Methode: `G.grid().reduce(GridClosureCallMode.SPREAD, P2, P3);`
Der erste Parameter weist an die Wörter an die Knoten zu verteilen. (Alternative *BROADCAST* sendet Daten an alle Knoten). *P2* gibt vor, welche data-collection verteilt werden soll (An diese Stelle kommt der Rückgabe-Wert der F.*yield*-Methode). *P3* steht dafür welcher Reduce-Typ ausgewählt wird (hier F.*sumIntReducer*()). Es folgt eine Beispielimplementierung eines *Wordcount*-Programms.

Quellcode

```
1.  realTimePopulate(final Grid g, CompletionService<Object> threadPool, final File inputDir)
2.          throws Exception {
3.          String[] books = inputDir.list();
4.          // Count closure which increments a count for a word on remote node.
5.          final GridClosure<Integer, Integer> cntClo = new GridClosure<Integer, Integer>() {
6.              @Override public Integer apply(Integer cnt) {
7.                  return cnt == null ? 1 : cnt + 1;
8.              }
9.          };
10.         final GridDataLoader<String, Integer> ldr = g.dataLoader(null);
11.         // Set larger per-node buffer size since our state is relatively small.
12.         ldr.perNodeBufferSize(300);
13.         for (final String name : books) {
14.             threadPool.submit(new Callable<Object>() {
15.                 @Override public Object call() throws Exception {
16.                     X.println(">>> Storing all words from book in data grid: " + name);
17.                     BufferedReader in = new BufferedReader(new FileReader(new
    File(inputDir, name)));
18.
19.                     try {
20.                         for (String line = in.readLine(); line != null; line =
    in.readLine())
21.                             for (final String w : line.split("[^a-zA-Z0-9]"))
22.                                 if (!w.isEmpty())
23.                                     ldr.addData(w, cntClo);
24.                     }
25.                     finally {
26.                         in.close();
27.                     }
28.
29.                     X.println(">>> Finished storing all words from book in data grid: " +
    name);
30.
31.                     return null;
32.                 }
33.             });
34.         }
35.
36.         int idx = 0;
37.
38.         while (++idx <= books.length) {
39.             try {
40.                 threadPool.take().get();
41.             }
42.             catch (Exception e) {
43.                 e.printStackTrace();
44.             }
45.         }
46.
47.         ldr.close(false);
48.     }
```

4.5.3 Ausführen von MapReduce-Aufträgen

In *GridGain* ist keine weitere Vorarbeit zur Ausführung eines Auftrags nötig, es reicht, die erstellte Java-Klasse einfach auszuführen, vorausgesetzt die benötigten Bibliotheken und Umgebungsvariablen sind korrekt eingebunden. Werden mehrere Knoten erwünscht müssen diese zudem bereits gestartet sein. *GridGain* unterstützt die Programmiersprachen *Java*, *Scala* und *Groovy* zur Erstellung von MapReduce-Aufträge. *Scala* und *Groovy* müssen zusätzlich installiert und konfiguriert werden, dies stellt allerdings keine größere Hürde dar. Entwickelt kann dann mit jeder der drei oder einer beliebigen Kombination der Sprachen. Sind die *GridGain* Bibliotheken korrekt eingebunden und der Code der Anwendung valide, so reicht es das Programm auszuführen. Es ist ebenfalls möglich die Aufträge in ausführbare Dateien wie z.B. ins JAR-Format zu exportieren und diese dann auszuführen. Aufträge werden an ein sogenanntes *Grid* übergeben welches dann je nach implementierter APIs/SPIs und Konfiguration, wie vorgegeben verhält, beispielsweise eine parallele oder zeitgenaue Ausführung. Unter anderem können den Aufträgen Prioritäten zugewiesen werden oder es kann konfiguriert werden wie sich die Knoten bei Ausfall oder Konflikten verhalten sollen. Neue Aufträge werden in eine *Job-Queue* eingefügt und diverse APIs/SPIs kümmern sich um die Verarbeitung. Die hierfür wichtige *GridCollisionSpi* hat die Aufgabe anstehende Aufträge zu aktivieren oder abzulehnen oder laufende Aufträge abzubrechen. Dabei wird diese aufgerufen, sobald neue Aufträge ankommen, fertig verarbeitet wurden oder die SPI aktiv aufgerufen wurde.
GridGain liefert eine Vielzahl von verschiedenen *Reduce*-Funktionen mit welche unterschiedliche Typen von Werte-Paaren zurückgegeben, auch ist es möglich, eigene Funktionen zu implementieren und Einfluss auf die Form der Rückgabe-Werte zu nehmen. Dabei können die Ergebnisse ganz normal weiterverwendet werden. Laut Hersteller sind die Knoten untereinander auch mit unterschiedlichen Betriebssystemen kompatibel, dies konnten wir aufgrund technischer Limitierungen allerdings nicht überprüfen.

4.5.4 Skalierung

Um *neue Knoten hinzuzufügen*, führt man die *ggstart.bat* im bin-dir von *GridGain* aus. Weitere Knoten können auf diese Weise ebenfalls *hinzugefügt werden* und erkennen sich gegenseitig, per *topology snapshot* werden in der Konsole die momentan laufenden Knoten protokolliert. Knoten können problemlos *während der Laufzeit hinzugefügt werden*, mit der gleichen Methode wie oben beschrieben. Das *Grid* erkennt den neuen Knoten umgehend und bindet ihn in die Verarbeitung der Aufgabe ein. Ein *Schließen von Knoten* gibt zunächst eine Fehlermeldung in der Konsole aus, woraufhin erkannt wird, dass ein Knoten entfernt wurde.
Der Hersteller macht über die *Grenzen der Skalierbarkeit* keine genauen Angaben, spricht jedoch auf der Internetseite in einem FeatureFlash von einer "Live Data Scaling to 1000s of Nodes" [GGF].
Die mitgelieferte *GridJobStealingCollisionSpi* unterstützt das Übertragen bzw. "Stehlen" von Aufträgen eines ausgelasteten Knoten zu einem nicht-ausgelasteten. Diese muss zusammen mit der *GridJobStealingFailoverSpi* verwendet werden, welche wiederum sicherstellt, dass ein Job auch wirklich reibungsfrei übergeben wird.

4.5.5 Dateisystem

GridGain unterstützt diverse verteilte Dateisysteme wie *Apaches HDFS* oder auch SQL-Datenbanken. Allerdings basiert die Software auf einer Verarbeitung von Daten, welche konstant im Speicher (in-memory) gehalten werden, daher beschränkt man sich bei der Arbeit mit *GridGain* auf das einmalige Einlesen von und Schreiben auf diese Dateisysteme, und der Hersteller weist

auch darauf hin, bei der Entwicklung der MapReduce-Aufträge auf diese Konzepte zu achten.
Weiterhin werden Schnittstellen zu Verfügung gestellt, welche das Verteilen von Aufträgen aus unterschiedlichen Quellen wie FTP oder HTTP zu ermöglichen. Da bei *GridGain* mit den Programmiersprachen *Java*/*Scala*/*Groovy* gearbeitet wird, kann bei Dateisystemoperationen auf die bekannten Befehle, in der Regel Java, zugegriffen werden. Mitgeliefert werden auch diverse Schnittstellen, welche den Zugriff auf verteilte Dateisysteme, Datenbanken oder Protokolle ermöglichen. *GridGain* stellt außerdem sicher, dass nur konsistente Daten auf die Knoten geschrieben werden können, bei Abweichungen in der Datenintegrität wird der Schreibzugriff verweigert. Dabei prüfen die einzelnen Knoten unabhängig gegen die Daten der anderen, je nach Einstellung in der Konfiguration. Der sogenannte "Segmentierungs"-Check wird von jedem Knoten an unterschiedlichen Punkten in der Verarbeitung durchgeführt, beispielsweise wenn sich *Grid*-Topologie verändert oder nach einer voreingestellten Zeitspanne. Auch das Verhalten beim Auffinden inkorrekter Daten-Segmente lässt sich per *GridSegmentationPolicy* konfigurieren. So ist es etwa möglich betreffende Knoten zu stoppen und aus dem Grid auszuschließen, oder eine vom Benutzer implementierte Funktion auszuführen um das Ereignis zu behandeln.

4.5.6 Fehlertoleranz

Es gibt viele Konfigurationsmöglichkeiten, wie sich *GridGain* bei *Ausfall eines Knotens* verhält (Aufgabe verwerfen, weitergeben, neu starten etc.). So können etwa Aufgaben automatisch an einen anderen Knoten übergeben werden. Bei *GridGain* lässt sich das Verhalten bei Ausfall unter bestimmten Bedingungen, beispielsweise dem Resultat des Auftrags, frei konfigurieren. Hierfür ist die *Failover SPI* zuständig. Außerdem ist es möglich, mithilfe der *Checkpint SPI* "Checkpoints" zu implementieren die bei langen Berechnungen Zwischenstände bzw. ganze Auftrags-Zustände an den oder einen Masterknoten schicken können damit bei Ausfall nicht von vorne begonnen muss. Falls der Masterknoten selbst ausfällt kann das Endergebnis per Checkpoint gespeichert werden und diesem erneut zugeschickt werden, sobald er wieder läuft.

4.5.7 Monitoring

GridGain bietet in seiner Enterprise-Variante grundsätzlich zwei verschiedene Möglichkeiten des Monitoring. Die erste Möglichkeit ist das *GridGain Visor*, dieses ist Konsolenbasiert und bietet eine Vielzahl von Befehlen um sich den Gesundheitsstatus von Knoten, Cache- oder Auslastungsstatistiken, oder auch aktuelle Konfigurationen anzeigen zu lassen. Dabei ist es auch möglich während der Laufzeit Knoten zu starten oder zu beenden, ebenso wie Benutzer-definierte Events zu definieren, welche beispielsweise eine E-Mail an den Admin senden können, falls die individuellen Bedingungen erfüllt sind.
Die *VisorUI* stellt die GUI-Alternative dar, welche auf denselben Scripten wie die Konsolenversion basiert. Es ist hier möglich bequem zwischen verschiedene Konfigurationen des Monitorings umzuschalten, welche in einer XML-Datei gespeichert werden. Das Programm bietet eine Vielzahl von Monitoringmöglichkeiten, etwa die Chart-unterstützte Überwachung von Knoten, Auslastung, Job-Status und mehr. In *Abbildung 4.5.7.1* sieht man das *VisorUI*-Dashboard in der Übersicht.

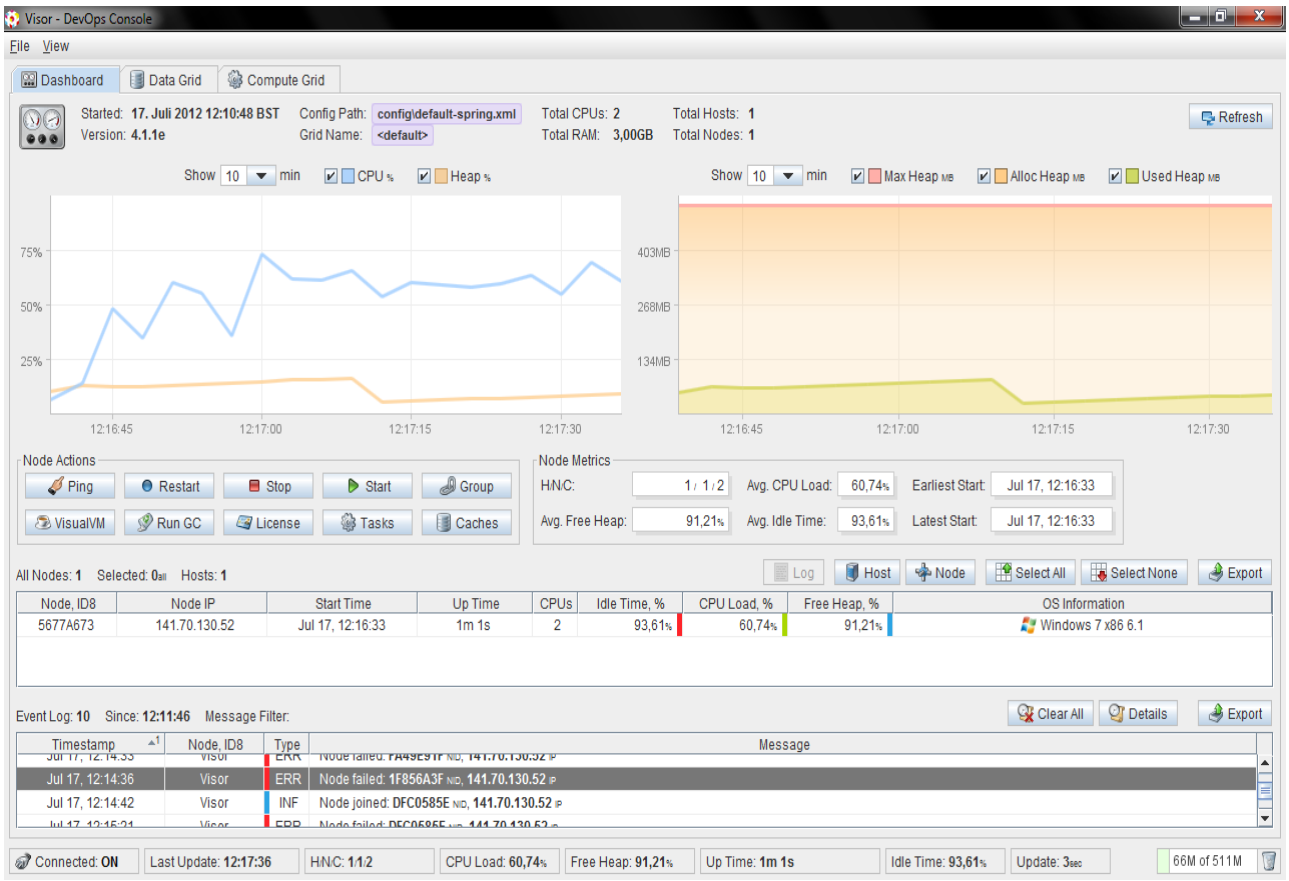

Abbildung 4.5.7.1: VisorUI-Dashboard

4.5.8 Fazit

GridGain lässt sich relativ einfach installieren und einrichten, solange Grundkenntnisse im verwendeten Betriebssystem und der Arbeit mit IDEs vorhanden sind. Die *Dokumentation* ist erfreulich ausführlich und gut erklärt, das Online-Portal bietet viele zusätzliche Informationen und Ressourcen die beim Einstieg und dem Einsatz der Software helfen. Das Erstellen und Ausführen von MapReduce-Auftrags ist mit ausreichenden Java-Kenntnissen ebenfalls machbar, der sich automatisch-skalierende Cluster, bei *GridGain* "*grid*" genannt, gibt zudem Feedback über den Gesundheitsstatus des Clusters sowie den Status der Auftrags-Ausführung. Zudem werden viele APIs und SPIs mitgeliefert welche das Ausfallen von Knoten, Performanzschwankungen, Lastbalancierung oder ähnlichen Konzepten behandeln. Die Konfigurationsmöglichkeiten sind groß, die Flexibilität der Software, auch durch die Unterstützung mehrerer Programmiersprachen und Betriebssysteme, ist ein klarer Pluspunkt. Das Monitoring ist nach unterer Einschätzung sehr gelungen. Schwankungen im *Grid* sowie Zustände der aktuellen MapReduce-Aufträge werden übersichtlich angezeigt und Konfigurationen lassen sich bequem wechseln.
GridGain ist nach unseren Erfahrungen für den produktiven Einsatz tauglich, bei unseren Testfällen trafen wir auf beinahe keine Probleme, und wenn, dann fand sich in der Dokumentation oder im gut besuchten Forum meist eine Lösung dazu.

4.6 Basho Riak

Basho Riak ist die erste Implementierung in der Auswahl der näher betrachteten Implementierungen, welche zunächst als eine Datenbank konzipiert und im Nachhinein mit einer

MapReduce-Funktionalität ausgestattet wurde. Wie gut diese umgesetzt wurde und ob das Gesamtpaket sogar interessanter sein könnte, wird sich in der nachfolgenden Untersuchung zeigen.

4.6.1 Installation, Inbetriebnahme und Dokumentation

Riak existiert in drei Versionen. Die Open-Source-Variante nennt sich einfach *Riak*, die kostenpflichtige Version *Riak Enterprise* und die Cloud-Version mit Unterstützung für Amazons S3 erhält den Namen *Riak CS*. Um *Riak* direkt zu installieren, hat man die Wahl zwischen vorkompilierten Installationspaketen für die Linux-Betriebssysteme Debian/Ubuntu, RHEL/CentOS, SUSE und für Mac OS X. Alternativ kann man *Riak* auch selbst kompilieren und installieren. Für MapReduce-Aufträge unterstützt *Riak* in der Basisversion die Sprachen Erlang, Javascript und ihre eigene HTTP-Schnittstelle, auf Wunsch können aber auch weitere Sprach-Bibliotheken für C/C++, Java, PHP, Python und Ruby installiert werden. Obwohl es fertige Installationspakete für *Ubuntu* gibt, stellt sich heraus, dass alle Tutorials und Konfigurationseinstellungen auf einer selbstkompilierten *Installation* basieren, so dass das Umkonfigurieren des fertigen Pakets umständlicher für den ersten Einstieg ist. U.a. wird mit dem fertigen Paket nur ein Cluster-Knoten erstellt und weitere Knoten müssen einzeln kopiert und konfiguriert werden.
Um ein *Riak*-Cluster bestehend aus mehreren Knoten aufzusetzen, ist es nicht vonnöten auch mehrere physikalische Knoten zu Verfügung zu stellen, sondern es steht einem frei mehrere virtuelle Knoten auf dem selben Knoten zu installieren und zu einem Cluster zu verbinden. Unabhängig davon, wie man sich entscheidet, erfordert die Installation von *Riak* eine Menge anderer Softwarekomponenten und stellte sich als recht kompliziert heraus. U.a. müssen alle Komponenten aus der Testsystem-Tabellenzeile "Zusätzliche Software" des *Abschnitts 4.2* genau in der angegebenen Reihenfolge installiert werden. Da diese auch nicht immer als fertige Binaries existierten, wie z.B. Erlang, mussten diese auch zuerst kompiliert und anschließend installiert werden bevor die eigentlich untersuchte *Riak*-Software kompiliert und installiert werden konnte.
Die *Dokumentation* unterstützt die Installation, welche ausschließlich über Linux-Kommandozeilen Befehle durchgeführt wird, teilweise sogar mit Videos, allerdings ist das Dokument an sich sehr unübersichtlich. Zum einen beziehen sich manche Installationsteile auf ältere Versionen und man muss einige Parameter selbst anpassen. Zum anderen wird irreführend als einzige Anforderung Erlang erwähnt, doch um Erlang installieren zu können, müssen wiederum andere Komponenten installiert werden, so dass man zu Beginn mit viele Fehlermeldungen konfrontiert wird. Insgesamt gestaltet sich die Installationsanleitung nicht intuitiv, da viele rekursive Schritte zu weiteren externen Anleitungen nötig sind, führt aber letztendlich zum Ziel. Im Zuge des Einstiegstutorials kann man sich einem Skript bedienen, welches auf der Software *Rebar* baut und mehrere Erlangprogramme, wie *Riak*, kopiert und automatisch für den Clusterbetrieb konfiguriert. Manuell kann der Ordner *Riak* im Installationsverzeichnis dupliziert werden und jeweils durch die Anpassung der Datei app.config für die Anbindung zu einem Cluster konfiguriert werden.
Die *Inbetriebnahme* jedes einzelnen Knotens, welches aus der Ordnerstruktur *"devX/bin/riak"* besteht, X steht für die Knotennummer, geschieht mittels des Konsolenbefehls: *"bin/riak start"*.
Die Vereinigung der einzelnen Knoten zu einem Cluster geschieht mittels des *Riak-Admin*-Befehls: *"devX/bin/riak-admin join dev1@<ip-des-ersten-nodes>".* Dieser Befehl muss für alle X Knoten wiederholt werden. Anders als die Installation verlief die *Inbetriebnahme* reibungslos, könnte aber bei sehr großen Clustern umständlich sein, da es kein fertiges Skript gibt, welches alle Knoten startet. Andere Implementierungen wie *Hadoop* und *MapR* sind diesbezüglich aber nicht besser, daher ist das nicht direkt als Nachteil zu bewerten.

Die *Dokumentation* kann online auf der *Riak*-Wiki-Seite eingesehen werden und besteht aus allgemeinen *Download*-und *Installationsanleitungen*, einem Einsteiger-Tutorial namens *Riak Fast Track*, einem Hauptteil, der die Funktionen, Konzepte, Einrichtung und Bedienung sehr detailliert beschreibt und einem Mailing-List-Archiv. Darüber hinaus können Informationen über ein Forum, Blog-Einträge von Fans und den Github-Commits bezogen werden. Obwohl einige Bereiche sogar mit Videos bereichert wurden, erschien es aus Nutzersicht so, als ob diese *Riak* verschönert darstellen würden. Notwendige Details um das Gezeigte im Video umsetzen zu können, sind teilweise auf vielen Seiten verteilt und erfordern ein tieferes Verständnis der Teilkomponenten und deren Bedienung. Aus diesem Grund wirkt die *Dokumentation* sehr professionell und vollständig und spielt einen schnellen Einstieg und eine leichte Bedienung vor, verlangt letztendlich aber eine längere Einarbeitungszeit.

4.6.2 Erstellung von MapReduce-Funktionen

Obwohl *Riak* primär ein verteiltes Datenbankensystem ist, das Daten innerhalb von *Buckets* als *Schlüssel* und *Wert* speichert, erlaubt es auch dem Google-MapReduce-Paradigma entsprechend die Abfrage und Verarbeitung dieser *Buckets*. Als Eingabe für die *Map*-Phase wird eine Liste von *Bucket-Schlüssel*-Paaren erwartet unabhängig davon, ob sie einen Wert haben oder nicht. *Riak* ermittelt automatisch die Datenpartitionen, die die *Bucket*-Schlüssel enthalten und fragt den zugehörigen *vNode* (= virtueller Riak-Knoten) ab, ob die *Map*-Funktion den Anforderungen entspricht und durchgeführt werden kann. Die *Reduce*-Phase hingegen akzeptiert als Eingabe eine beliebige Liste von Daten und produziert als Ausgabe auch eine beliebige Liste von Daten. Diese werden allerdings erst einmal nur temporär gespeichert und über die Konsole ausgegeben. Der Nutzer muss also bei Wunsch einer dauerhaften Speicherung der Ergebnisse die Funktion um entsprechende Codezeilen anpassen. Im Folgenden sieht man beispielhaft, wie ein MapReduce-Auftrag aussehen kann. Zuerst werden Textzeilen in dem Bucket "alice" unter den Schlüsseln "p1", "p2" und "p5" gespeichert und anschließend mittels der *Riak* eigenen HTTP-Schnittstelle über die Konsole durch zwei JavaScript- Funktionen bearbeitet.

Quellcode (Wörterzählen)

```
1            {"inputs":[["alice","p1"],["alice","p2"],["alice","p5"]]
2     ,"query":[{"map":{"language":"javascript","source":"
3     function(v) {
4       var m = v.values[0].data.toLowerCase().match(/\w*/g);
5       var r = [];
6       for(var i in m) {
7         if(m[i] != '') {
8           var o = {};
9           o[m[i]] = 1;
10          r.push(o);
11        }
12      }
13      return r;
14    }
15    "}},{"reduce":{"language":"javascript","source":"
16    function(v) {
17      var r = {};
18      for(var i in v) {
19        for(var w in v[i]) {
20          if(w in r) r[w] += v[i][w];
21          else r[w] = v[i][w];
22        }
23      }
24      return [r];
25    }"}}]}
```

Zeile 1 beschreibt die Eingabe-Buckets, Zeile 2 und Zeile 15 initiieren jeweils die *Map*- und *Reduce*-Funktion und definieren dabei ihre Sprache. In Zeile 3 bis 14 findet man die *Map*-Funktion, welche eine Liste aus JSON-Objekten erstellt, bei dem jedes Wort den Schlüssel darstellt und als Wert eine "1" erhält. In Zeile 16 bis 25 befindet sich die Reduce-Funktion, welche jedes JSON-Objekt in der Liste betrachtet, zu jedem Schlüssel ein neues Objekt erstellt, welches entsprechend der Vorkommensmenge des Wortes je ein Schlüssel erhält, dass dann anschließend gezählt und als ein neues JSON-Objekt mit dem Wort und dessen Anzahl gespeichert wird.

4.6.3 Ausführen von MapReduce-Aufträgen

Riak unterstützt eine hohe Anzahl an Sprachen um entsprechende Aufträge ausführen zu können. Da es selbst auf Erlang basiert, werden diese mit nativer Geschwindigkeit unterstützt. Javascript-Aufträge werden über die SpiderMonkey-Engine, welche in *Riak* integriert ist, übersetzt und nach Aussage des Herstellers mit vergleichbarer Geschwindigkeit ausgeführt. Weitere Clients für C/C++, Java, PHP, Python und Ruby können nachinstalliert und ebenso genutzt werden. Bevor MR-Aufträge ausgeführt werden können, müssen die entsprechenden Daten innerhalb von Bucket-Schlüssel-Paaren gespeichert vorliegen. *Riak* stellt dazu einige Skripts zur Verfügung, die auch gewöhnliche Dateien in ein Cluster importieren und entsprechend aufbereiten. Leider gibt es derer nur sehr wenige und diese müssen auch noch entsprechend der Datei manuell angepasst werden. Als nächstes ist die Bedienung der Linux-Shell notwendig und mit folgender Befehlszeile wird ein Auftrag eingeleitet: *curl -X POST http://<Cluster-IP-Adresse>:<Portnummer>/mapred -H "Content-Type: application/json" -d @-*
Curl unterstützt das HTTP-Protokoll und erlaubt das Einfügen, Entfernen und Auslesen von Daten. Um mit POST die MapReduce-Ergebnisse in der Konsole darzustellen, ist es notwendig den Datentyp als JSON-Objekte zu definieren. Nach dem @- kann eine MapReduce-Funktion analog zum Beispiel unter *Punkt 4.1* direkt in die Konsole geschrieben, oder wenn sie als eine ausführbare Datei vorliegt auch als solche samt Dateipfad aufgerufen werden. *Riak* bietet aus sich heraus keine *Job-Queue*, so dass diese sequentiell angestoßen werden müssen. Es gibt aber Open-Source-Erweiterungen, die sich diesem Mangel annehmen wollen, wie z.B. *Riaque* eines französischen Teams. Alle Ein- und Ausgaben während der *Map*- und *Reduce*-Phasen werden bis zur endgültigen Ausgabe zwischengespeichert und können während dieser Zeit beliebig weiterverwendet werden. So ist es auch möglich mehrere *Map*- und anschließend mehrere *Reduce*-Funktionen hintereinander auszuführen, solange die Ein- und Ausgabedatentypen den Anforderungen der entsprechenden Phasen (siehe Abschnitt 4.6.2) genügen. Da *Riak* im Grunde nur Unix-Systeme unterstützt und entsprechende Knoten mittels derselben Befehlszeilen zu einem Cluster zusammengefügt werden, ist davon auszugehen, dass unterschiedliche Linuxdistributionen/Mac OSX miteinander harmonieren sollten. Mangels Vielfalt unserer Testsetups und eines Mac-Betriebssystems, konnten wir dieses Verhalten aber nicht testen.

4.6.4 Skalierung

Das *Hinzufügen eines virtuellen oder physischen Knotens* geschieht immer auf die selbe Art und Weise, da beide nur als Ordnerstrukturen inklusive einer Konfigurationsdatei samt IP-Adresse vorliegen und als solche identifiziert werden. In der Linuxkonsole wird wie unter *Abschnitt 4.6.1* beschrieben, die Befehlszeile *devX/bin/riak-admin join dev1@<ip-des-ersten-nodes>* für den Knoten X, der dem Cluster hinzugefügt werden will, ausgeführt. Alle Knoten werden immer dem ersten Knoten, hier *dev1/bin/riak*, hinzugefügt. Mit der Befehlszeile *curl -H "Accept: text/plain" http://<Cluster-IP-Adresse>:<Portnummer>/stats* können die Details des Clusters eingesehen

werden, u.a. etwa, ob der neue Knoten erfolgreich hinzugefügt wurde und seinen Anteil der Datenpartitionen übernommen hat. Beide Funktionen können überdies durch die grafische Oberfläche umgesetzt werden, welche unter *Abschnitt 4.6.8* erläutert wird.
Da bereits für das Einrichten des Clusters zuerst alle Knoten gestartet werden müssen und sie während des Betriebs nacheinander zusammengefügt werden, kann auf selbe Art und Weise jederzeit ein weiter *Knoten zur Laufzeit hinzugefügt werden*. Die Integration in bereits laufende Clusterprozesse, wie MapReduce-Aufträge, werden dynamisch an neue Knoten mit ausgelagert. Ähnlich dem *Hinzufügen eines Knotens* kann man statt dem "riak-admin"-Befehl *join* einfach *leave* eintippen und schon verlässt dieser den Cluster und seine Datenpartitionen werden automatisch unter den übriggebliebenen Knoten verteilt. Falls der zu entfernende Knoten nicht reagiert, weil ein Soft- bzw. Hardwarefehler vorliegt, kann dieser mit dem "riak-admin"-Befehl *force-remove* entfernt werden, auch wenn dadurch Datenreplikationen, die auf dem Knoten lagen, verloren gehen. Falls das *Entfernen eines Knotens zur Laufzeit* nicht erzwungen werden musste, werden alle Datenreplikationen auf die anderen Knoten im Cluster umgeleitet und der zu entfernende Knoten sicher abgekoppelt.

4.6.5 Lastbalancierung

Riak unterstützt zwei Lastbalancierungs-Strategien. Eine nennt sich *virtual IPs,* die andere *reverse-proxy*. Zu beiden Strategien gibt es viele Implementierungen, die von *Riak* akzeptiert werden. Von Haus aus verteilt *Riak* die Arbeitslast sowie Datenreplikationen voll automatisch, so dass alle Knoten im Cluster ausgelastet werden. In Fällen, in denen es sinnvoller ist MapReduce-Aufträge nur in der Nähe der physikalisch liegenden Daten auszuführen, wird dies erkannt und entsprechend umgesetzt. Interne Beobachtungen bestätigen dieses Verhalten.

4.6.6 Dateisystem

Das Importieren von systemnativen Daten in ein *Riak*-Cluster geschieht über eine eigene HTTP-Schnittstelle mittels der Linuxkonsole. Mit dem Befehl *curl -X PUT HTTP://<Cluster-IP-Adresse>:<Portnummer>/riak/images/on_cluster.jpg \ -H "Content-type: image/jpeg" --data-binary @input.jpg* wird z.B. eine JPEG-Bilddatei mit dem Namen "*input*" als Bilddatei "*on_cluster.jpg*" im Cluster gespeichert. Notwendige Parameter sind zum einen der *PUT*-Befehl, zum anderen die Cluster-Adresse und der Speicherpfad und ganz wichtig der Dateityp hinter dem *Content-type* Schlüsselwort. Da die Daten nun über den Webbrowser unter der angegebenen Speicheradresse betrachtet werden können, ist es über diesen ebenfalls möglich die Dateien auf das systemnative Dateisystem "herunterzuladen". *Riak* unterstützt eine große Anzahl von sogenannten *Storage-Backends*, die je nach Anforderungen an das Datenbankensystem ausgewählt werden können. *Bitcask* ist eine Erlangapplikation, die eine Schnittstelle für das Speichern und Laden von Schlüssel/Wert-Daten in einer Hashtabelle erlaubt, welche verteilt auf dem Hauptspeicher aller Knoten im Cluster liegt und damit eine geringe Latenz und insgesamt hohe Datendurchsätze bieten soll. Es ist das Standart-Storage-Modul von *Riak*. *Innostore* ist ebenso eine Erlangapplikation und bietet ein verlässliches, transaktionales Speichersystem für Riak an, welches dem von MySQL gleicht. *LevelDB* wurde von Google-Mitarbeitern entwickelt und ist ein Neuling unter den "Schlüssel/Wert"-Datenbanken. Seine Architektur entspricht mehr dem von Googles *BigTable* und hat daher nicht die RAM-Limitierung von *Bitcask*. Aus dem Grund sieht *Riak LevelDB* als einen idealen Partner für die eigene Software an. *Multi* steht für multiple Storage-Backends und erlaubt das kombinierte Benützen von z.B. *Bitcask* für schnelle Operationen auf kleinen Datensätzen und *LevelDB* für das langfristige Speichern und Verarbeiten von Daten.

4.6.7 Datenkonsistenz und Fehlertoleranz

Aufgrund *Riaks* Datenbank-Spezialisierung, besitzt es mehrere Strategien um die Daten im Cluster zu sichern. Mittels Replikationen können Daten entweder gleichmäßig im Cluster verteilt werden oder auf Wunsch nur auf bestimmten Knoten und Clustern als Read-Only-Daten zur Verfügung gestellt werden. Da es keinen Masternode gibt und auf jedem Knoten *Riak* vollständig installiert und mit Ausführungsrecht vorliegt, existiert der bekannte *SPoF* nicht. Sobald ein Knoten ausfällt, werden benachbarte Knoten informiert und die Anfragen und Aufgaben des ausgefallenen Knotens werden von diesen übernommen. Auf gleiche Weise werden die fehlenden Datenreplikationen auf anderen Knoten automatisch erzeugt. Dieser Wert wird zu Beginn der Clustererstellung bestimmt und liegt standardmäßig bei 2 Replikationen. Nach Aussagen von *Basho* entstehen bei Ausfall eines Knotens kaum Geschwindigkeitseinbußen und es wird stets das Lesen als auch das Schreiben von Daten gewährleistet. Aufgrund der geringen Ressourcen konnten wir diese Aussage allerdings nicht überprüfen.

4.6.8 Monitoring

Obwohl der Status eines *Riak*-Clusters per Konsolenbefehl abgefragt werden kann, ist dessen Textausgabe unübersichtlich strukturiert und einige wichtige Informationen erfordern weitere Konsolenbefehle. Mit *Riak Control* hat man eine webbasierte grafische Oberfläche geschaffen, die zum Zeitpunkt der Fachstudie eine Übersicht über den Gesundheitsstatus des gesamten Clusters gibt, den Status jedes einzelnen Knotens inkl. Partitionsgröße und Speicherplatzverbrauch anzeigt und Informationen über die "Ring"-Partitionszugehörigkeit gibt (siehe *Abschnitt 4.6.9 Sonstiges*).

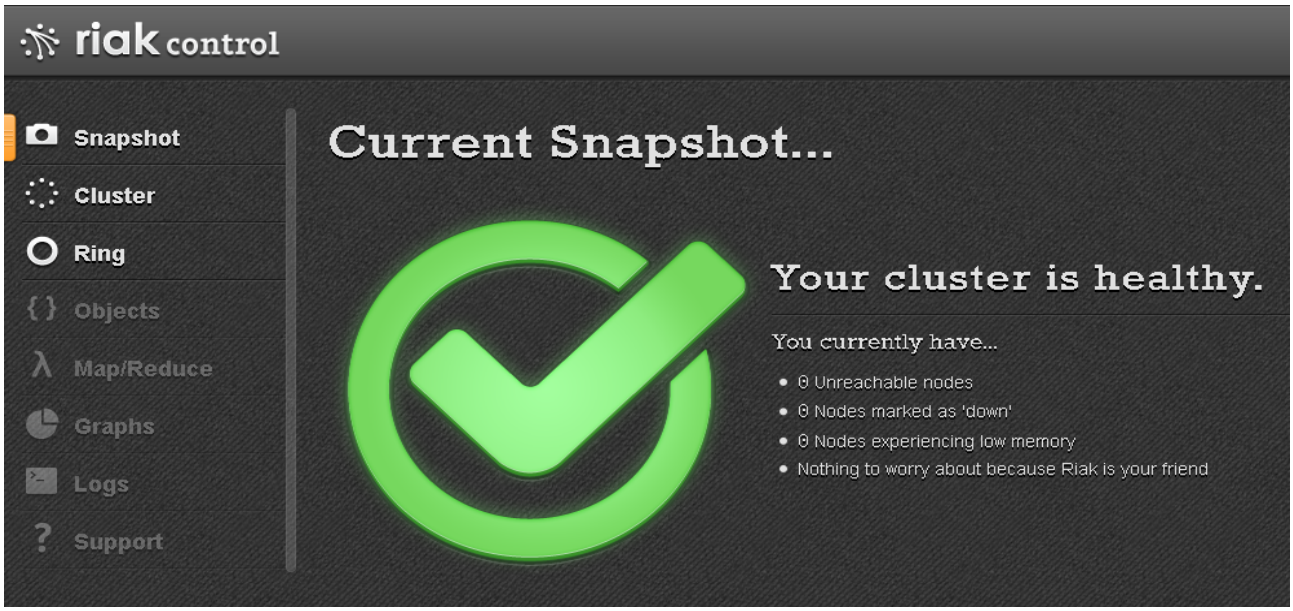

Abbildung 4.6.8.1: Startseite von Riak Control

Nach der Installation und Konfiguration von *Riak Control*, kann man in einen Webbrowser die selbst bestimmte HTTPS-Adresse eingeben und nach Eingabe eines Benutzernamens und Passworts gelangt man in die in der oberen *Abbildung 4.6.8.1* dargestellten Ansicht. Diese erlaubt in der linken Leiste die Navigation zu verschiedenen Ansichten des Clusters, welche auf der rechten Seite dargestellt werden. In diesem Fall erhalten wir Informationen über den Gesundheitsstatus des gesamten Clusters, welcher vorerst in Ordnung zu sein scheint. Sobald man im linken Bereich auf "Cluster" klickt, erscheint die Ansicht welche auf *Abbildung 4.6.8.2* zu sehen ist.

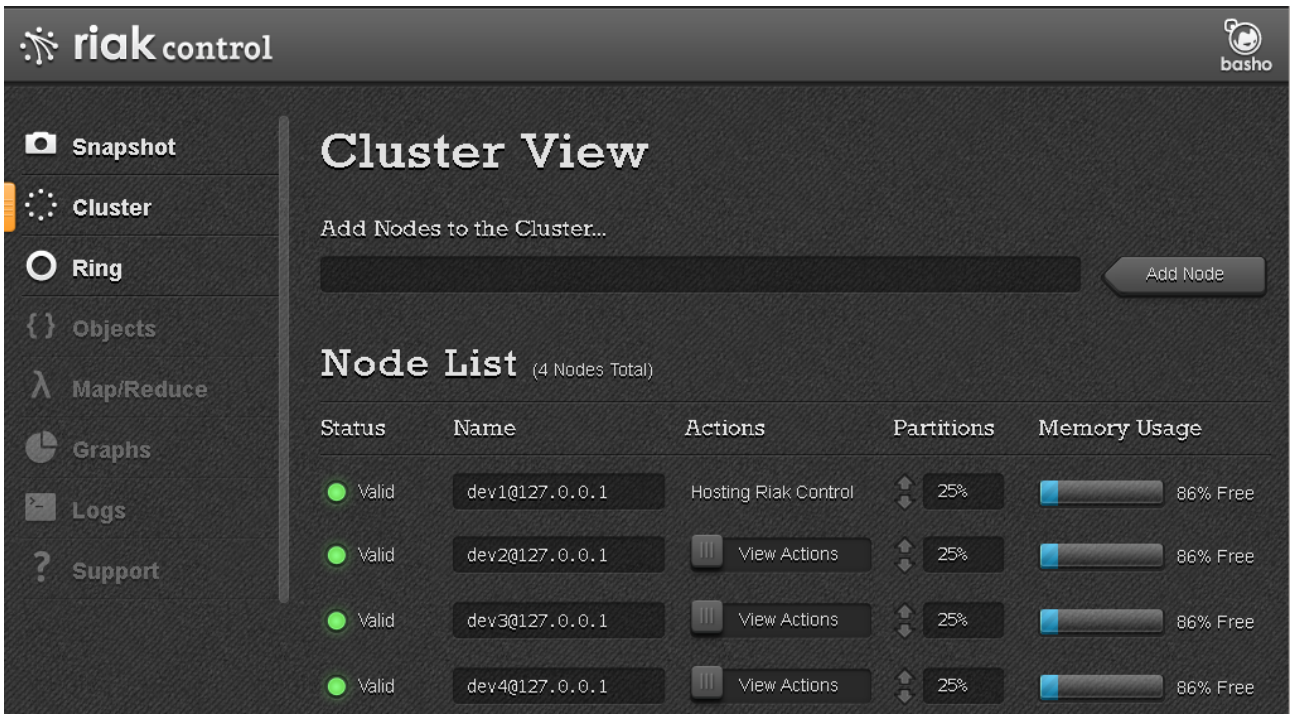

Abbildung 4.6.8.2: Übersicht der Knoten in einem Cluster

Entsprechend unseres Testsetups werden alle vier Knoten inkl. ihrer Funktionsfähigkeit, der Partitionsgröße und des freien Speicherplatzes in Tabellenform aufgelistet. Auf Wunsch können weitere Knoten in den Cluster mit dem gleichen Konsolenbefehl - wie unter *Abschnitt 4.6.4* beschrieben - hinzugefügt werden.

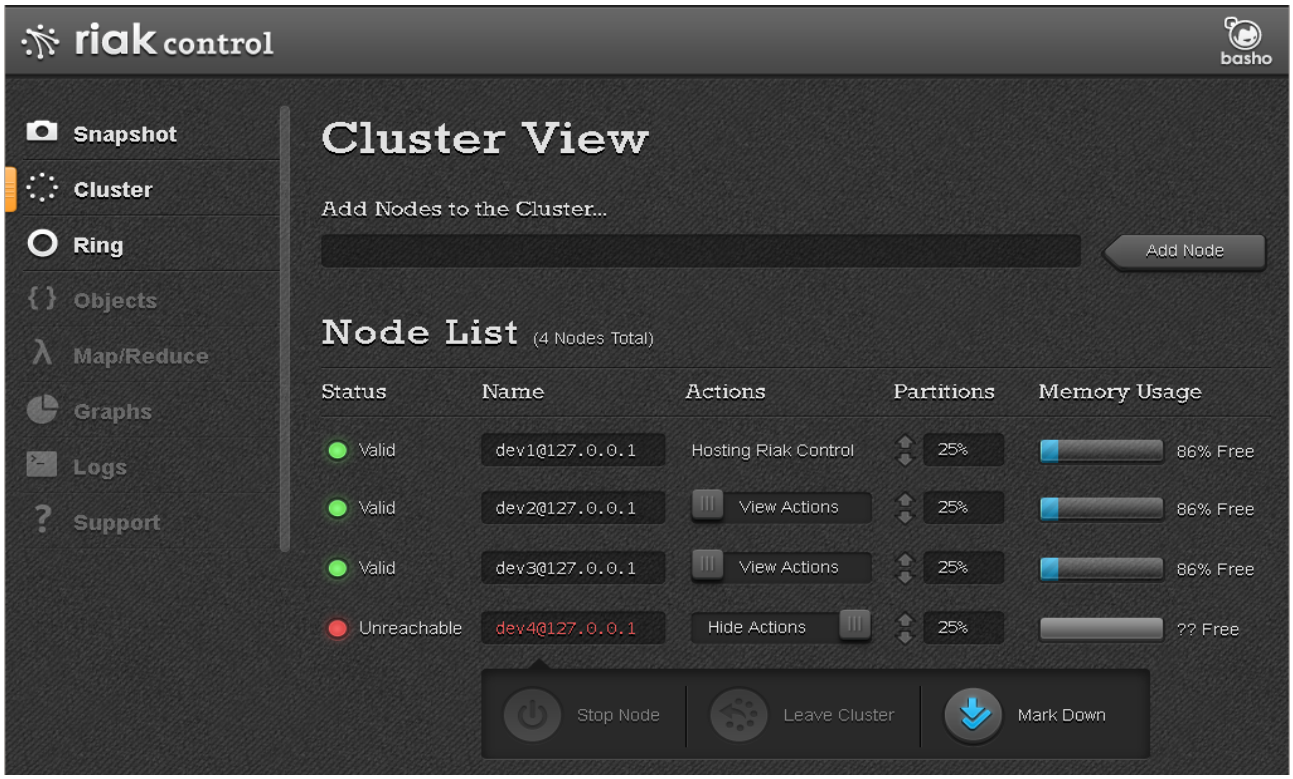

Abbildung 4.6.8.3: Unerreichbare Knoten werden rot markiert

Durch einen Schieberegler hat man die Wahl auch weitere Aktions-Buttons für jeden einzelnen Knoten einzublenden, mit denen man diesen stoppen oder aus dem Cluster entfernen kann. Gestoppte Knoten oder aufgrund eines Fehlers nicht mehr erreichbare Knoten können als "Down" markiert werden, um z.B. fehlerhafte Hardware auszutauschen oder den Knoten für andere Zwecke zu nutzen (*Abbildung 4.6.8.3*). Unabhängig von der Ursache des Fehlers, wird in der *Snapshot*-Übersicht ein Fehler im Cluster gemeldet und visuell deutlich dargestellt, wie in *Abbildung 4.6.8.4* zu sehen ist.
Auch wenn *Riak Control* noch am Anfang seiner Entwicklung steht und viele Funktionen, wie in der linken Leiste ausgegraut zu sehen, noch nicht verfügbar sind, macht es einen stabilen Eindruck und lässt sich über den Browser problemlos bedienen. Zukünftig soll man die Datenobjekte (*Buckets*) nicht nur sehen, sondern auch über die Oberfläche bearbeiten können, was bisher nur sehr umständlich über die Konsole realisierbar ist. Auch soll es später möglich sein, MapReduce-Aufträge sowohl zu erstellen, zu starten als auch zu überwachen. Diverse Graphen-Darstellungen und Logdateien und ein direkt erreichbarer Support gehören ebenfalls zu den weiteren Plänen für *Riak Control*.

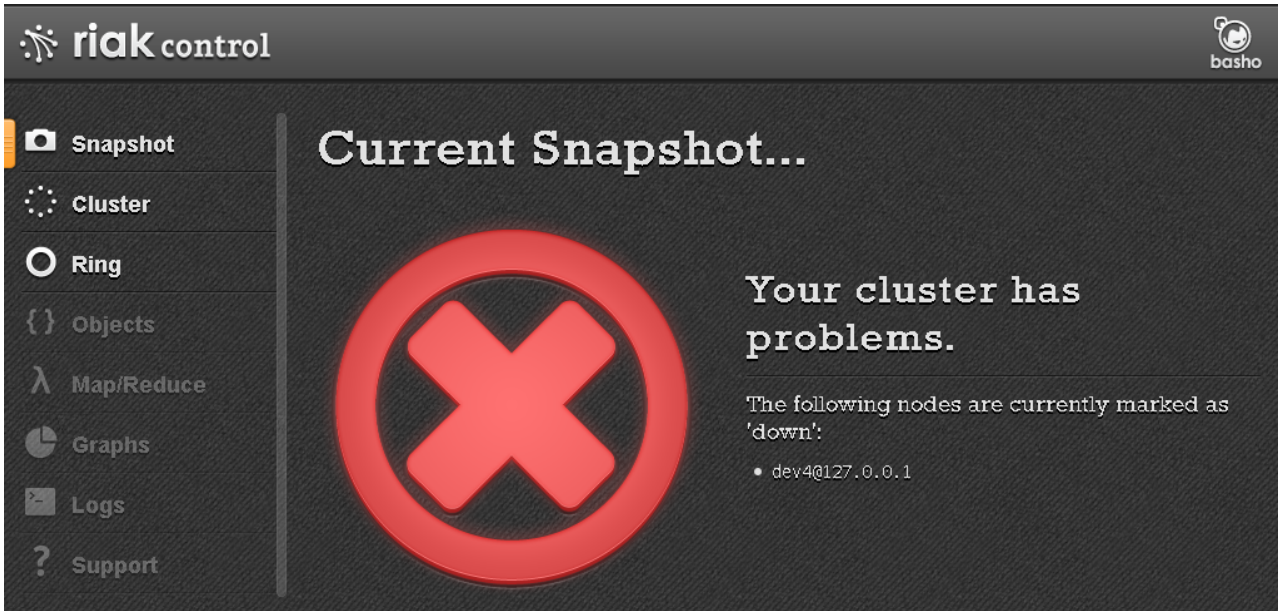

Abbildung 4.6.8.4: Fehlerdarstellung im Cluster

4.6.9 Sonstiges

Erwähnenswert ist noch das *Riak* Ring-Prinzip, welches alle *Buckets* in einem Cluster auf einen 160-Bit-Integerraum projiziert, der als ein Ring dargestellt wird und entsprechend der Knotenanzahl in Partitionen aufgeteilt wird [RIR]. Die Partitionszahl entspricht dabei einem Vielfachen der Knotenzahl und kann selbst konfiguriert werden. *Riak* splittet jeden Knoten in sogenannte "vnodes" (virtual nodes) und teilt jede Partition einem dieser *vnodes* zu, so dass jeder Knoten eine Vielzahl unterschiedlicher Partitionen besitzt und auf diese Weise eine Datenredundanz auch auf Prozessebene gewährleistet ist und nicht nur auf Knotenebene.

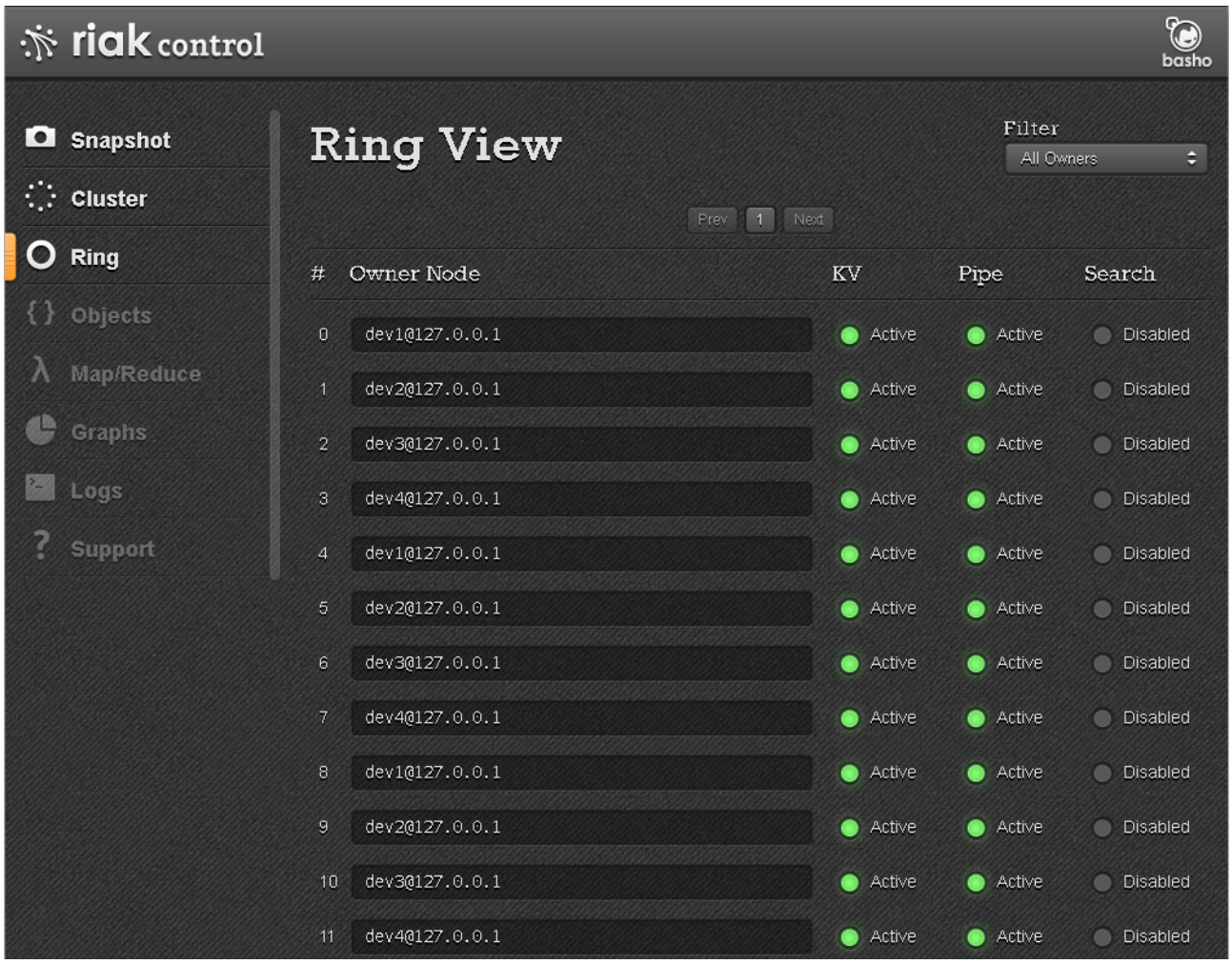

Abbildung 4.6.9.1: Aufteilung der Ringpartitionen auf die einzelnen Knoten

Die *vnodes* und ihre sogenannte Ring-Zugehörigkeit kann, wie man in *Abbildung 4.6.9.1* sieht, mittels *Riak Control* auch überwacht werden. Jede Ringpartition erhält ihren eigenen festen Indexwert und der Reihe nach erhält jeder echte Knoten eine dieser Partitionen. *Riak KV* und *Riak Pipe* sind zwei verschiedene MapReduce-Techniken, deren Verfügbarkeit in je einer Spalte gezeigt wird. *Riak Search* ist ein Indizierungssystem, welches das schnelle Finden von Textstellen in den *Buckets* ermöglicht. Da es in *KV* schon integriert ist, ist es in unserem Testcluster auch nicht aktiviert gewesen. Da die Anzahl der Ringpartitionen bei größeren Clustern sehr hoch wird, können diese nach diversen Parametern, wie man in *Abbildung 4.6.9.2* sehen kann, gefiltert werden.

Abbildung 4.6.9.2: Ringansicht mit gewähltem Filter

Falls man einen kürzlich errichteten *Riak*-Cluster auf seine Leistungsfähigkeit überprüfen will, bietet sich mit *Basho Bench* eine Software an, die unterschiedliche Testmodi bietet und eventuelle Flaschenhälse sichtbar machen kann. Da viele *Riak*-User ihre Ergebnisse teilen und mögliche Fehlerquellen ergründen, entsteht so die Möglichkeit die eigenen Werte mit derjenigen der Nutzerbasis zu vergleichen und dadurch keine Leistung bei einem *Riak* Setup zu verschenken.
Insgesamt ist die Nutzerbasis von *Riak* im Forum und der Mailinglist sehr aktiv und entwickelt in vielen Nebenprojekten, die von Github gehostet werden, Erweiterungen um u. a. fertige MapReduce-Funktionen für verschiedene Aufgabenstellungen zur Verfügung zu stellen und das für alle Sprachen, die *Riak* mit den entsprechenden Clients unterstützt. Weiterhin versuchen einige Fans weitere Sprach-Clients zu entwickeln, die ihnen fehlen, so dass man z.B. auch mit C# und .Net MapReduce-Funktionen erstellen kann. Auf die gleiche Weise war *Riak Control* zunächst ein Fanprojekt und wurde irgendwann in das offizielle Repository von *Riak* hinzugefügt.

4.6.10 Fazit

Wenn man sich zum ersten Mal mit *Basho* befasst, hat man den Eindruck einer professionellen Softwarefirma, die ihre Produkte aufgrund eines hübschen Webauftritts mit vielen Grafiken und Videos gut zu präsentieren weiß. *Riak*, welches ihr Hauptprodukt darstellt, ist zum einen eine NoSQL-Datenbanksoftware, die sich auf eine leichte Skalierung der Clustergröße und der fortwährenden Verfügbarkeit für Lese- und Schreiboperation mittels virtueller Knoten und Replikation spezialisiert hat. Zum anderen verspricht *Riak* die Abfrage und Verarbeitung der Daten über eine eigene Umsetzung des MapReduce-Paradigmas, das von Google entworfen wurde.
Nach unserer im Rahmen einer Fachstudie möglichen, tiefer gehenden Untersuchung können wir bestätigen, dass die Umsetzung der Datenbankfunktion gut gelungen ist, da z.B. neue Knoten sehr leicht erzeugt und in ein Cluster eingebunden werden können, sei es über die Konsole, oder über die *Riak Control* genannte, grafische Benutzeroberfläche. Ebenso sind einmal in den Cluster eingefügte Daten auch bei Ausfall eines Knotens jederzeit verfügbar. Nicht sehr positiv ist allerdings der Aufwand, der nötig ist, um Daten in den Cluster zu importieren, da jede Datei inklusive ihres

Dateityps, der Speicheradresse und des Dateinamens umständlich in die Konsole eingegeben werden musste. Firmen, die oft unterschiedliche Datentypen und große Mengen in den Cluster einfügen wollen, werden zwangsweise ihre eigenen Skripte schreiben müssen. Gleiches gilt für den Export, welcher die Kenntnis der exakten Dateiadresse erfordert, um diese mit einem Browser navigieren und herunterladen zu können. Zukünftig sei geplant, beide Funktionen über die Web-GUI zu realisieren, die in ihrem jetzigen Entwicklungsstand noch unvollständig ist, aber mit ihrer übersichtlichen Clusterüberwachung und der intuitiven Bedienung bereits einen ordentlichen Eindruck macht. Bezüglich der MapReduce-Funktionalität stellte sich heraus, dass *Riak* viele Programmiersprachen unterstützt und entsprechend erstellte Aufträge sehr zuverlässig ausgeführt wurden, dennoch vermisst man u.a. eine *Job-Queue*, als auch die Möglichkeit Jobergebnisse ohne großen Programmieraufwand in einer Datei abspeichern zu können, denn diese werden standardmäßig nur in der Konsole dargestellt und entsprechend fehlt auch eine Logdatei.
Positiv hervorzuheben sind die vielen Programmerweiterungen der Nutzerbasis, welche nicht nur vier unterschiedliche Storage-Backends für die Datenspeicherung bieten, von denen nur *Bitcask* als Standardwahl getestet wurde, sondern auch weitere Programmiersprachen für MapReduce-Funktionen umfassen. Am Beispiel von *Riak Control* ist auch zu sehen, dass Riak eng mit den Nutzern zusammenarbeitet, auf Fragen und Wünsche eingeht und gute Erweiterungen in ihre Hauptsoftware integriert. Trotz der unübersichtlich aufgebauten *Dokumentation* und den daraus entstehenden Einstiegsschwierigkeiten, scheint *Riak* v.a. aufgrund ihres engagierten Entwickler- und Supporteams von namhafte Hersteller wie AOL und Mozilla genutzt zu werden. Daher könnte *Riak* unserem Eindruck nach auch im Bereich der MapReduce-Implementierungen zu einer anerkannten Größe werden, wenn diese wie versprochen irgendwann in der Weboberfläche direkt ausführbar und überwachbar sind und Clusterdateien leichter zugänglich werden.

4.7 MongoDB

Das NoSQL-Datenbanksystem ist interessant wegen seiner JavaScript-basierten MapReduce-Auftrag Erstellung, sowie der vermeintlich einfachen Installation und Verwendung. Im Folgenden möchten wir dies näher untersuchen.

4.7.1 Installation, Inbetriebnahme und Dokumentation

Die *Installation* von *MongoDB* beschränkt sich auf das Entpacken der Programmdateien und das Zuweisen eines Ordners für die Datenhaltung. Zusätzliche Software ist nicht notwendig.
Es ist keine Konfiguration vonnöten. Nach dem Start des Serverprozesses und der Zuweisung der Knoten des Clusters über simple Konsolenbefehle ist MongoDB vollständig lauffähig.
Es ist zwar eine Vielzahl von Konfigurationsmöglichkeiten vorhanden, jedoch dienen diese größtenteils der Leistungsoptimierung oder der Erstellung komplexer Clusterarchitekturen.
Die Dokumentation von *MongoDB* ist sowohl detailliert als auch umfangreich. Während des gesamten Installations- und Testprozesses war sie stets in der Lage, eventuelle Verständnisprobleme zu lösen.

4.7.2 Erstellung von MapReduce Funktionen

MongoDB implementiert MapReduce-Funktionalität als Datenbankkommando. Dieses erwartet in Javascript implementierte *Map*- und *Reduce*funktionen als Eingabeparameter.
MapReduce arbeitet unter *MongoDB* stets auf einer Collection, dem NoSQL-Pendant einer Tabelle.

Die *MongoDB*-Shell ist JavaScript-basiert. Folglich lassen sich MapReduce-Funktionen nur in JavaScript implementieren. Unter *MongoDB* ist die MapReduce-Funktinalität als Datenbankkommando implementiert. Dieses kann entweder über die Konsole, oder aus einem Skript heraus gestartet werden. Im Folgenden wird die Implementierung einer MapReduce-Funktion veranschaulicht. Als Grundlage dient hier das klassische *Wordcount*-Beispiel.

Quelltext

```
1    var map = function() {
2            var words = this.text.split(" ");
3            for (var wordNum in words)
4            {
5                    emit(words[wordNum], 1);
6            }
7    }
8
9    var reduce = function(key, values) {
10       var wordCount = 0;
11       values.forEach(function(value) {
12               wordCount += value;
13               });
14
15       return wordCount;
16   }
17
18   var MapReduceOperation = db.test.mapReduce(map, reduce, {out: "mr_results"});
```

In der Map-Funktion ist auffällig, dass der *this* Pointer in *MongoDB* MapReduce anders verwendet wird, als in konventionellem JavaScript. Er bezeichnet hier das aktuell bearbeitete Dokument der Collection. In *Zeile 18* ist das Datenbankkommando *mapReduce* zu sehen. Die Eingabeparameter sind die Map-Funktion, die Reduce-Funktion, und die Ausgabecollection.

4.7.3 Ausführen von MapReduce-Aufträgen

Da MapReduce unter *MongoDB* als Datenbankkommando implementiert ist, und somit nur innerhalb von *MongoDB* als Skript ausgeführt wird, ist keinerlei Vorarbeit notwendig. Weil das System zudem prinzipiell eine Datenbank darstellt, besitzt es keine *Job-Queue* als solchen. Es existieren jedoch Datenbanktreiber für Python, Java, Ruby, PHP und Perl. Über diese können Datenbankkommandos, und somit auch MapReduce-Aufträge, an einen *MongoDB*-Cluster übergeben werden. Auf die Ergebnisse eines MapReduce-Aufträge kann unter *MongoDB* auf zweierlei Weise zugegriffen werden. Einerseits über die spezifizierte Ausgabecollection, andererseits über den Job selbst. Dieser wird als JavaScript-Objekt gehalten, welcher die Ausgabe des Auftrags als Eigenschaft besitzt. Das Erstellen eines Clusters aus mehreren Betriebssystemen ist problemlos möglich. Ein gleichzeitiges verwenden von x86 und x64 Architekturen ist jedoch nicht möglich.

4.7.4 Skalierung

Knoten können mithilfe eines einzelnen Datenbankkommandos (*addshard*) *hinzugefügt werden*. Soll ein ReplicaSet (mehrere identische Knoten) hinzugefügt werden, genügt es, einen der Repliken hinzuzufügen. Da *MongoDB* nur über Datenbankkommandos konfiguriert werden kann, besteht keine andere Möglichkeit, als *Knoten zur Laufzeit hinzuzufügen*. *Das Entfernen eines Knotens* ist über ein Datenbankkommando (*removeshard*) möglich. Dieses muss zwei Mal übergeben werden. Das erste Mal, um die Datenmigration auf andere Knoten zu beginnen, das zweite Mal, um den Knoten endgültig zu entfernen. Sollte das Kommando ein zweites Mal gegeben werden, bevor die

Datenmigrierung abgeschlossen ist, wird der Status der Migration angezeigt. Wie das Hinzufügen von Knoten ist auch ein Entfernen selbiger nur zur Laufzeit über die Konsole möglich. *MongoDB* ist zum Zeitpunkt der Fachstudie mit bis zu 100 Knoten getestet. Langfristiges Ziel der Entwicklung ist die Unterstützung von bis zu 1000 Knoten.

4.7.5 Lastbalancierung

MongoDB besitzt zwei Funktionalitäten, welche zum Zwecke der Lastbalancierung benutzt werden können: *Sharding* und *ReplicaSets*. Beim *Sharding* werden einzelne Collections anhand eines Shardkeys auf mehrere Chunks aufgeteilt, welche auf verschiedenen Knoten (*Shards*) gespeichert werden. Mithilfe eines in Anbetracht des Anwendungsszenarios angemessen Shardkeys lässt sich der Schreib-/Rechenaufwand auf mehrere Knoten verteilen (*Write Balancing*). Unter *ReplicaSets* versteht man eine Menge von redundanten Knoten, welche dieselben Daten beinhalten. Da dieselben Daten von allen Knoten gelesen werden können, verringert sich die Leselast pro Knoten (*Read Balancing*).

4.7.6 Dateisystem

MongoDB speichert Daten in Form von BSON-Dokumenten, einem JSON-Derivat. Neue Daten können über das Datenbankkommando *insert* gespeichert werden. In diesem Fall wird das native Dateisystem des Hostbetriebssystems genutzt. BSON-Dokumente sind jedoch in ihrer Größe begrenzt (4MB). Um auch größere Dateien speichern zu können, besitzt *MongoDB* eine Datenspezifikation namens GridFS, welche den Zugriff auf mehrere Dokumente als eine Datei unterstützt. Um GridFS zu nutzen, stehen APIs für PHP, Java, Python, Ruby und Perl zur Verfügung. Über diese lassen sich Daten bequem zwischen nativem Dateisystem und *MongoDB* übertragen. Da *MongoDB* direkt auf das Dateisystem des Hostbetriebssystems zugreift, unterstützt es sämtliche Dateisysteme, welche von kompatiblen Betriebssystemen unterstützt werden. Da *MongoDB* eine Datenbank darstellt, sind Zugriffe auf die Daten nur über Datenbankkommandos möglich. *MongoDB* unterstützt, wie in Abschnitt 4.1 erwähnt so genannte *ReplicaSets*. Diese bestehen aus Slave- und Masterknoten. Fällt ein Slave aus, hat dies keine Auswirkungen auf das Verhalten des Clusters. Fällt der Masterknoten eines *ReplicaSets* aus, so wird von den verbleibenden Knoten automatisch ein neuer Master bestimmt. Fällt ein Knoten aus, welcher nicht Teil eines *ReplicaSets* ist, gehen dessen Daten verloren. Bei der Betrachtung der Auswirkungen einer hohen Knotenanzahl muss auf Grund der Architektur von *MongoDB* zwischen Zwei Fällen unterschieden werden: Einer hohen Anzahl von Repliken pro *ReplicaSet*, und einer hohen Anzahl einzelner Shards. Die Vorteile der Erhöhung dieser Entitäten wurde bereits in *Abschnitt 4.7.4* betrachtet.

4.7.7 Fehlertoleranz

Da bei *MongoDB* Implementierung und Dateisystem eine Einheit bilden, ist dieser Abschnitt in Anbetracht von *Abschnitt 4.7.6* redundant.

4.7.8 Monitoring

MongoDB besitzt mehrere integrierte Monitoring-Tools, sowie eine Vielzahl an Monitoringmöglichkeiten von Drittanbietern. Die Implementierung bietet außerdem ein spartanisches WebUI, welches grundlegende Metriken (Uptime, Anzahl Queries etc.), einen Datenbankbrowser, so wie Zugriff auf den Datenbanklog bietet. Die Metriken sind eine Teilmenge der durch *mongostat* erfassbaren Daten. *Mongostat* ist ein konsolenbasiertes Diagnosetool,

welches mit einem beliebigen *mongod*-Prozess verbinden kann. Die ausgegebenen Metriken sind umfangreich, beziehen sich allerdings nur auf einen einzelnen Knoten. *Mongotop* ist eine konsolenbasierte Anwendung, welche Informationen über Lese- und Schreibaktivitäten auf Datenbanken und einzelnen Collections anbietet. Dabei werden nur solche Collections/Datenbanken angezeigt, welche in einem spezifizierten Zeitraum Aktivität aufweisen. *Mongotop* überwacht nur einzelne Knoten. Zusätzlich zu den integrierten MongoDB-Monitoringtools, existiert noch eine Reihe von Programmen/Plugins von Drittanbietern:

- *Munin*: Serverstatistiken, Collectionstatistiken
- *Ganglia-Plugin*: Ähnliche Metriken wie Mongostat zur Visualisierung mit Ganglia
- *Cacti*: Grafische Anzeige von Serverstatisitken
- *Motop*: Wie *Mongotop*, parallele Verbindung zu mehreren Servern möglich
- *Mongo Live*: Chrome-Plugin zur Knotenüberwachung

4.7.9 Fazit

Auf den ersten Blick ist bei der Benutzung von *MongoDB* auffällig, dass es sich hierbei in erster Linie um eine NoSQL-Datenbank handelt, und die MapReduce-Funktionalität eher Beiwerk darstellt. Dies wird vor allem daran deutlich, dass MapReduce-typische Funktionalitäten, wie zu Beispiel ein Checkpointing-System für die Wiederaufnahme von Aufträgen nach einem Knotenausfall nicht implementiert sind. Des weiteren lassen sich MapReduce-Aufträge bei allen Monitoringoptionen nicht separat erfassen, die Bearbeitung von Datenbanken und einzelnen Datensätzen steht deutlich im Vordergrund. *MongoDB* besitzt jedoch auch seine Vorteile: Der Konfigurations- und Installationsaufwand ist minimal, MapReduce-Aufträge lassen sich einfach und ohne Einbinden externer Bibliotheken implementieren und die Möglichkeiten zur Datenmanipulation gehen weit über die Möglichkeiten reiner MapReduce-Frameworks hinaus. Falls MapReduce in einer Produktionsumgebung nicht das Kernstück der Infrastruktur, sondern nur einen von mehreren Datenmanipulationsansätzen darstellt, ist *MongoDB* definitiv einen Blick wert.

4.8 MySpace Qizmt

4.8.1 Installation, Inbetriebnahme und Dokumentation

Qizmt kommt in einer kleinen Datei und die Installation ist sehr simpel gestaltet. Die einzige Hürde dabei ist, dass beim Setzen des "Service Logins" zusätzlich zum Benutzername des OS noch der Name der Maschine angegeben werden muss, beispielsweise "computerXY\BenutzerA". Der Hinweis darauf fehlt in der von uns getesteten Version. Die Installationsanleitung und -beispiele sind ansonsten geradlinig verfasst, sind verständlich und es funktioniert wie beschrieben.

4.8.2 Erstellung von MapReduce-Funktionen

Die Erstellung erfolgt über eine beliebige Systemkonsole und dem Befehl `Qizmt edit <Dateiname des neuen Auftrags>`. Da für gewöhnlich mit XML-Code gearbeitet wird, bietet sich die Endung .xml an, z.B. *WordCount.xml.* Nach Bestätigung des Befehls wird der *Qizmt Jobs Editor* geöffnet indem man den Code einfügen und abspeichern kann, vgl. *Abbildung 4.8.2.1.*

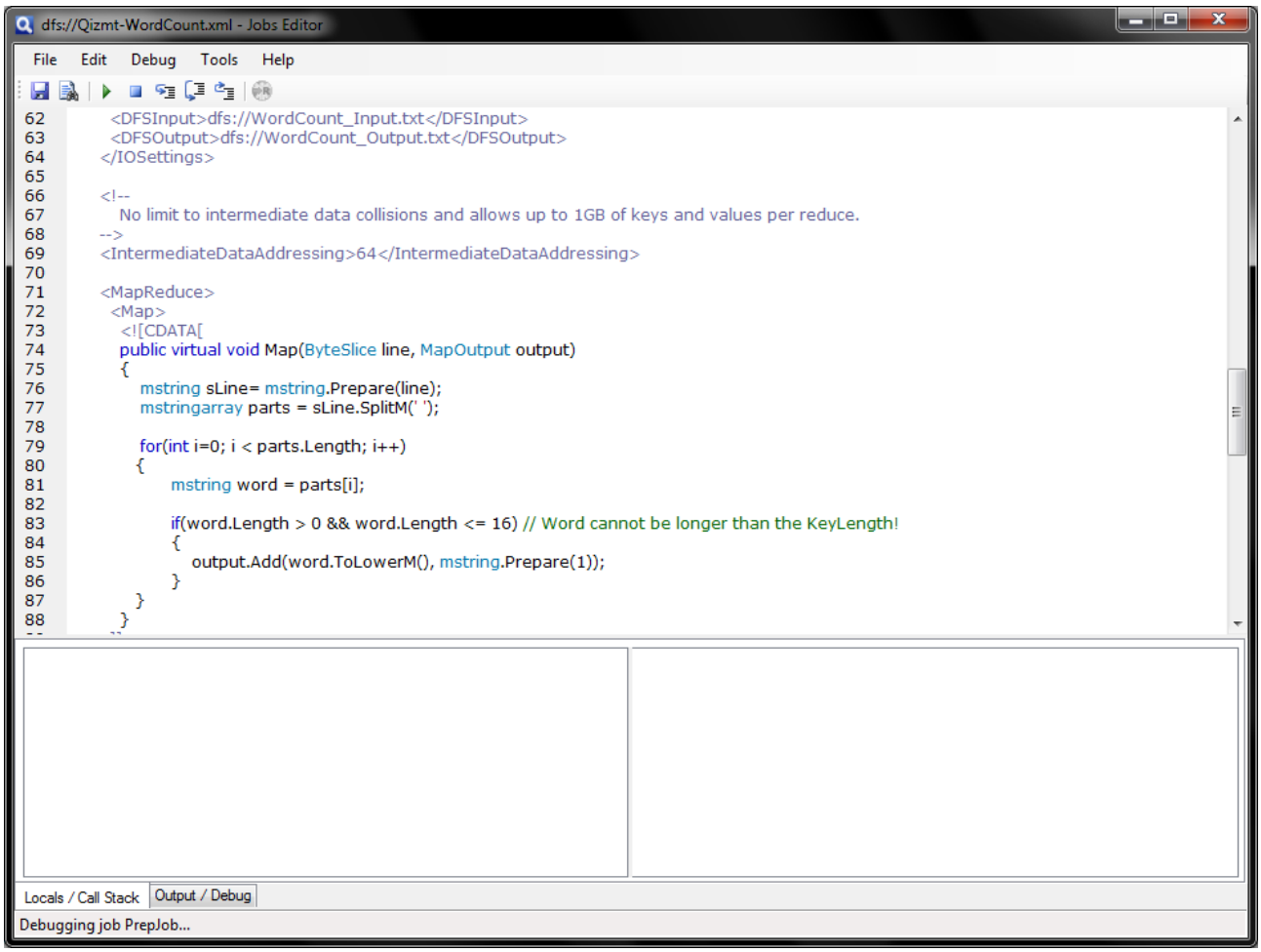

Abbildung 4.8.2.1: Qizmt Job Editor

Innerhalb eines MapReduce-Auftrag-Dokuments können ein oder mehrere Aufträge angelegt werden. Die Aufträge werden dabei in der Reihenfolge ausgeführt, wie sie im Dokument angeordnet sind. Es gibt drei Typen von Aufträgen, die im <JobType>-Tag in der XML-Datei festgelegt werden:

local: wird meist auf einer Maschine (lokal) ausgeführt, beispielsweise um Ergebnisse zu veröffentlichen.

remote: wird auf einer oder mehreren Remote-Maschinen ausführt und basiert meist auf einer Datei aus dem Dateisystem um z.B. Daten im Cluster zu verteilen.

mapreduce: Aufgaben, die nach dem MapReduce-Paradigma für gewöhnlich an mehrere Knoten verteilt, und im Reduce-Schritt wieder zusammengeführt werden.

Innerhalb des Tags erfolgt die Konfiguration des Auftrags, außerdem werden Ein- und Ausgabepfade festgelegt. Im MAP-Tag des XML-Dokuments wird der Map-Schritt nach dem MapReduce-Paradigma in der Programmiersprache C# implementiert. Im REDUCE-Tag wird der Reduce-Schritt implementiert. Es folgt ein Beispiel einer MapReduce-Funktion per in XML eingebettetes C#, so wie es vom Hersteller mitgeliefert wird.

Quellcode (Zählen der Worthäufigkeit in einem Dokumentsatz)

```
<?xml version="1.0" encoding="utf-8"?>
<SourceCode>
  <Jobs>
    <Job Name="WordCount" Custodian="" email="">
      <IOSettings>
        <JobType>mapreduce</JobType>
        <KeyLength>16</KeyLength>
        <DFSInput>dfs://WordCount_Input.txt</DFSInput>
        <DFSOutput>dfs://WordCount_Output.txt</DFSOutput>
      </IOSettings>
      <MapReduce>
        <Map>
          <![CDATA[
          public virtual void Map(ByteSlice line, MapOutput output)
          {
              mstring sLine= mstring.Prepare(line);
              mstringarray parts = sLine.SplitM(' ');

              for(int i=0; i < parts.Length; i++)
             {
                    mstring word = parts[i];

                    if(word.Length > 0 && word.Length <= 16)
                    {
                        output.Add(word.ToLowerM(), mstring.Prepare(1));
                    }
             }
          }
        ]]>
        </Map>
        <Reduce>
          <![CDATA[
          bool HasSaved = false;
          List<byte> SavedKey = new List<byte>();
          int SavedCount = 0; // Init.
          void HandleSaved(ReduceOutput output)
          {
                if(HasSaved)
                {
                    mstring sLine = mstring.Prepare(ByteSlice.Prepare(SavedKey));
                    sLine = sLine.AppendM(',').AppendM(SavedCount);
                    output.Add(sLine);
                    HasSaved = false;
                    SavedCount = 0; // Reset.
                }
          }

          [KeyRepeatedEnabled]
          public override void Reduce(ByteSlice key, ByteSliceList values, ReduceOutput output)
          {
              if(!Qizmt_KeyRepeated)
              {
                  HandleSaved(output);
              }

              SavedKey.Clear();
              UnpadKey(key).AppendTo(SavedKey);
              SavedCount += values.Length;
              HasSaved = true;

              if(StaticGlobals.Qizmt_Last)
              {
                  HandleSaved(output);
              }
          }
        ]]>
        </Reduce>
      </MapReduce>
    </Job>
  </Jobs>
</SourceCode>
```

4.8.3 Ausführen von MapReduce-Aufträgen

MapReduce-Aufträge werden bei *Qizmt* in C# geschrieben, der Code wird jedoch in ein vorgegebenes XML-Grundgerüst verpackt.
Weiterhin ist es möglich eine SQL-Erweiterung zu installieren um so eine Interoperabilität zwischen den MapReduce-Aufträgen und einer Datenbank zu ermöglichen.
Qizmt besitzt einen eingebauten Editor über den Aufträge erstellt, bearbeitet und debugged werden können. Ausgeführt wird ein Auftrag über den Befehl *Qizmt exec <Dateiname des Auftrags>*. Dabei ist anzumerken, dass in *Qizmt* kein Master-Knoten existiert. Der Auftrag kann auf einem beliebigen Knoten ausgeführt werden. Dieser wird automatisch an die restlichen Knoten im Cluster verteilt.
Die Informationen über einen Auftrag in der *Queue* werden über den oben beschriebenen Auftrags-Tag zu Verfügung gestellt. Die *Qizmt* Command-Line stellt eine Vielzahl von Befehlen zu Verfügung um mit der *Job-Queue* zu interagieren, so können die momentan bearbeiteten Aufträge sowie die noch Anstehenden etwa mit dem Befehl ps angezeigt werden. Einen alternativen Zugriff, wie etwa per GUI, gibt es nicht. Ebenfalls besteht die Möglichkeit zum MapReduce-Caching bei der Zwischenergebnisse von MapReduce-Aufträgen für weitere Berechnungen verwendet werden können.

4.8.4 Skalierung

Per Konsolenbefehl *Qizmt addmachine <hostname>* bzw. *Qizmt remove machine <hostname>* können Knoten zum Cluster hinzugefügt oder entfernt werden. Der Cluster kann jedoch ebenfalls über den Befehl *Qizmt format machines=<hostname1>,<hostname2>,<...>* automatisch eingerichtet werden. Die Knoten werden dabei in die Datei slave.dat eingetragen, welche sich dann ggf. auch manuell manipulieren lässt.
Administrative Änderungen, unter anderem eine Änderung des Clusters während der Ausführung von Aufträgen, sind bei MySpace *Qizmt* nicht möglich. Zu den Grenzen der Skalierbarkeit von *Qizmt* haben wir keine Angaben vom Hersteller oder in der Dokumentation gefunden.

4.8.5 Dateisystem

Für den Zugriff auf Dateien des *Qizmt*-Dateisystems müssen die Dateien im XML-Tag <Job> wie gefolgt referenziert werden:

```
<Job …
 <IOSettings>
   .
   .
   .
  <DFSInput>dfs://testfile.txt</DFSInput>
```

bzw.

```
  <DFSReader>dfs://testfile.txt</DFSInput>
```

wenn ein Datenstream-Input benötigt wird.
Bei der Verarbeitung der Daten gibt es diverse Möglichkeiten, welche in der Dokumentation jedoch nicht ausreichend erklärt werden. Lediglich in den Beispielen und den FAQs findet man einige Hinweise auf eine mögliche Verarbeitung des Inputs sowie das Verwenden mehrerer Eingabestreams gleichzeitig.
Für das Schreiben von Dateien in das Qizmt-Dateisystem müssen die Dateien im XML-Tag <Job>

wie gefolgt referenziert werden:

```
<Job …
  <IOSettings>
    .
    .
    .
 <DFSOutput>dfs://testfile.txt</DFSInput>
```

bzw.

```
   <DFSReader>dfs://testfile.txt</DFSInput>
```

für Datenstream-Ausgabe. Exportiert kann dann mithilfe der Klassen *MapOutput/ReduceOutput* bzw. *RemoteOutputStream.* Nach unserem Kenntnisstand, basierend auf den von uns erstellten Testfällen sowie der in der Dokumentation vorhandenen Informationen, ist es nur möglich auf das *Qizmt*-eigene Dateisystem MR.DFS zuzugreifen. Laut Herstellerangaben beinhaltet *Qizmt* eine Daten-Redundanz, dies konnte von uns jedoch weder überprüft werden, noch konnten weitere Informationen dazu in der Dokumentation oder dem Internet gefunden werden [QIZ].

4.8.6 Fehlertoleranz

Das absichtliche Entfernen bzw. "Abschießen" eines Knotens hatte bei unseren Tests ein "Hängenbleiben" des Auftrags zur Folge. Laut Hersteller werden Aufträge an funktionierende Knoten als "Failover" übergeben sobald das "Redundancy"-Flag beim Formatieren des Clusters korrekt gesetzt und konfiguriert wird. Dies konnten wir aufgrund fehlender Informationen in der Dokumentation allerdings nicht einstellen.

4.8.7 Monitoring

Qizmt bietet einen Debugger mit dem selbst erstellte MapReduce-Aufträge Schritt für Schritt debugged werden können, um so Fehler während der Ausführung von Aufträgen aufzuspüren. Weitere Monitoring-Features konnten wir allerdings nicht finden. Auch die Dokumentation stellt zu diesem Thema keine weiteren Informationen zu Verfügung.

4.8.8 Fazit

Qizmt bietet eine schnelle und einfache Möglichkeit MapReduce-Aufträge zu erstellen und auszuführen. Die *Installation* und das *Einrichten* des Clusters ist in der Theorie sehr simpel gestaltet, auch wenn wir beim Testen auf größere Probleme mit den Zugriffsrechten auf die Konfigurations- und Clusterdateien im Netzwerk gestoßen sind. Die *Dokumentation* ist leider an vielen Stellen mangelhaft und wird ebenso wie die Software selbst seit einiger Zeit nicht mehr aktualisiert. Daher ist die Zukunftsaussicht von *Qizmt* leider sehr in Frage zu stellen, was vermutlich auch mit dem Schwächeln von MySpace zusammenhängt [TMS]. Letztendlich ist ein produktives Einsetzen von *Qizmt* aufgrund der aktuellen Probleme, der fehlenden Monitoring-Funktionen sowie der schwierigen Konfiguration aufgrund der lückenhaften Dokumentation, eher abzuraten. Möchte man jedoch lediglich in MapReduce herein schnuppern oder einen Prototypen eines MapReduce-Auftrags möglichst ohne viel Aufwand ausprobieren, ist *Qizmt* dennoch wegen des geringen Einrichtungs-Aufwands eine Alternative.

4.9 Übersicht über die Ergebnisse der detaillierten Betrachtung

Eine Übersicht über die Ergebnisse der detaillierten Betrachtung findet man in der Tabelle im *Anhang 3.1.* In der f*olgenden Abbildung 4.9.1* sieht man eine Beispielübersicht für ein paar Implementierungen zu einigen Betrachtungs-Aspekten.

Testkategorien	Apache Hadoop	Gridgain	MapR
MapReduce			
1. Erstellung			
1.1 Wortzahlermittlung	erfolgreich	erfolgreich	erfolgreich
1.3 Unterstützte Sprachen	Java (mit Streaming beliebige)	Java, Scala, Groovy	Java (mit Streaming beliebige)
1.4 Möglichkeiten zur Job-Erstellung/Änderung	Konsole/Theoretisch IDE-Unterstützung	per IDE	
2. Ausführung			
2.1 Nötige Vorarbeit	mittel	niedrig-mittel	sehr niedrig
2.2 Job Queue API	vorhanden	vorhanden	vorhanden
2.3 Weiterverwenden der Ergebnisse	möglich	möglich	möglich
2.4 Kompatibilität bei verschiedenen OS	kompatibel, nicht empfohlen	voll kompatibel **	Clients für OSX und Windows
3. Skalierung			
3.1 Knoten hinzufügen	manuelle Konfiguration	möglich	möglich
3.2 Knoten hinzufügen zur Laufzeit	möglich	möglich	möglich
3.3 Knoten entfernen	manuelle Konfiguration	möglich	möglich
3.4 Knoten entfernen zur Laufzeit	möglich	möglich	möglich
3.5 Skaliebarkeit auf hohe Anzahl von Knoten	Bis 2000 in Praxiseinsatz	möglich **	mehr als 10.000 **
3.6 Automatische Skalierbarkeit	nein	möglich	möglich
4. Lastbalancierung			
4.1 Vorhandensein	nur per Hbase	vorhanden	vorhanden
4.2 Konfigurationsmöglichkeiten	vorhanden	umfangreich	vorhanden

Abbildung 4.9.1: Beispiel-Ergebnisse aus der Detaillierten Betrachtung

5 Bewertung der Ergebnisse

5.1 Allgemein

Bei den von uns untersuchten Implementierungen haben sich im Großen und Ganzen drei Gruppen herausgebildet: Erstens eine Gruppe, welche das von Google eingeführte MapReduce-Framework lizenziert und darauf aufbaut. Hierzu gehören unter anderem Apache Hadoop mit seinen vielen kommerziellen Distributionen. Diese werden immer noch weiterentwickelt und sind weit verbreitet – vermutlich auch noch in absehbarer Zukunft.
Die zweite Gruppe besteht aus den Implementierung, die sich an der Originalimplementierung orientieren, allerdings eigene Algorithmen implementiert haben. Diese stellte in dem von uns gesetzten Rahmen die größte Gruppe dar und weitere, teilweise recht kleine Vertreter finden sich noch im Internet. Allerdings gibt es einen hohen Anteil an Projekten, die bereits inaktiv sind oder nicht mehr regelmäßig auf den aktuellen Stand gebracht werden wie etwa MySpace *Qizmt*, daher ist unklar, ob und wie die Zukunft dieser Gruppe aussieht, zumal einige Kandidaten für sehr spezielle Aufgabenbereiche entwickelt wurden.
Die dritte Gruppe der Implementierungen, welche zwar Möglichkeiten zum Nutzen von MapReduce-ähnlichen Konzepten erlauben, sich allerdings nicht darauf beschränken oder gar noch weiter gehen möchten, dürfte stark im kommen sein. Zwar gibt es noch nicht viele vergleichbare Alternativen und einige Kandidaten, wie zum Beispiel *Stratosphere* oder *Daytona* sind noch in der Entwicklungs- oder Testphase, durch die offenere und generalisierte konzeptuelle Einstellung bezüglich des MapReduce-Paradigmas können wir allerdings erwarten, dass diese nun häufiger behandelt werden und sich die erhöhte Flexibilität bei den künftigen Technologieeinführungen auszahlen wird.
Auch auf dem Gebiet der Verarbeitung von großen Datenmengen und der verteilten Berechnung entwickeln sich die Technologien und Konzepte immer weiter, daher wird es wohl auch in Zukunft Projekte geben, die sich den neuen und kommenden Techniken annehmen, wie zum Beispiel *Stratosphere*. Andere werden die aktuellen Konzepte und Technologien optimieren und ausreizen, wie zum Beispiel *MapR*. Dabei scheint es keinen klaren Trend in Richtung Open-Source oder proprietärer Software zu geben, vermutlich wird es in beiden Sektoren weiterhin neue Implementierungen geben.
Die Tendenz bezüglich des Auslieferungsumfangs geht nach unserer Einschätzung in die Richtung, dass die Basisversionen der Pakete kostenfrei zu Verfügung gestellt werden, welche die grundlegenden Funktionen von Cluster und MapReduce(-ähnlichen)-Aufträgen enthalten. Komfort- und Monitoringfunktionen, sowie Unterstützung von größeren Datenmengen und Clustergrößen sind dann in den kostenpflichtigen Versionen enthalten.
Java ist die Sprache auf die wir am häufigsten gestoßen sind, was vermutlich auf deren Hype der letzten Jahre zurückzuführen ist. Jedoch ist auffällig, dass bei den neueren bzw. aktuelleren Implementierungen mehr Sprachen unterstützt werden als bei den Übrigen. Dies legt nahe, dass der Trend hier definitiv in Richtung Flexibilität bezüglich der Erstellung von MapReduce-Aufträgen oder ähnlichen Berechnungen auf großen Datenmengen geht. Oftmals auch durch Unterstützung diverser Skriptsprachen wie z.B. *Scala*, welche eine kürzere und einfachere Auftrags-Erstellung ermöglichen.
Ansonsten kann man zu vielen Programmiersprachen die passende MapReduce-Implementierung finden, C#, Javascript oder Queries mit SQL gehören dazu.

5.2 Genau betrachtete Implementierungen

Das Erstellen von MapReduce-Funktionen bereitete bei keiner der betrachteten Implementierungen Probleme. Abgesehen vom Erstellen der jeweiligen *Map*- und *Reduce*-Funktionen war wenig bis kein zusätzlicher Entwicklungsaufwand notwendig, auch ließen sich die erstellten Programme problemlos ausführen.
Es zeichnete sich bei allen betrachteten Implementierungen eine Tendenz zur Unterstützung unterschiedlicher Sprachen aus. Dies reichte von voller Kompatibilität via STDIN/STDOUT bei *Hadoop* und *MapR* bis hin zu der einfachen Unterstützung mehrerer Skriptsprachen bei *GridGain* und Basho *Riak*. Nur *MongoDB* und Myspace *Qizmt* unterstützen nur eine einzige Sprache für das Erstellen von Aufträgen. Die im Rahmen der Fachstudie genauer betrachteten MapReduce-Implementierungen machten insgesamt einen ausgereiften Eindruck. Während der Testphase waren Systemabstürze eine äußerst seltene Erscheinung.
Das Ausführen von Aufträgen ist bei allen getesteten Implementierungen über die Systemkonsole möglich. Zusätzlich dazu bieten die *Hadoop*-basierten Implementierungen sowie *GridGain* die Möglichkeit, via IDE-Integration aus dem Funktionsfluss eines Programmes heraus auf die Funktionalität des Clusters zuzugreifen. Die beiden NoSQL-Datenbanken bieten mit der Fähigkeit, über Datenbanktreiber Queries zu starten, eine ähnliche Funktionalität.
Die für das Ausführen von MapReduce-Aufträgens nötige Vorarbeit ist ebenso durch die Bank gering. Die *Hadoop*-basierenden Implementierungen und *GridGain* benötigen lediglich das Einbinden einiger Java-Bibliotheken und einen Kompiliervorgang, bei *Qizmt* lassen sich Aufträge direkt und ohne Umwege aus dem mitgelieferten Editor starten. *Riak* und *MongoDB* können Aufträge direkt über Datenbankqueries starten, jedoch müssen die zu verarbeitenden Dateien zuerst in das jeweilige NoSQL-Datenbankformat konvertiert werden. Die Weiterverwendung der Ausgabe in nachfolgenden Aufträgen war bei allen betrachteten Implementierungen problemlos möglich, einziger Aufwand war bei den Hadoop-basierten Implementierung die Konvertierung der Datentypen.
Eine weitere Gemeinsamkeit der genauer betrachteten Implementierungen ist der angetroffene Konfigurations- und Bedienungsstil. Dieser richtet sich deutlich an die professionelle Anwendung auf Serverplattformen: GUIs und Wizards sind selten, Konfiguration über Konsole und Textdateien sind die Norm. *Hadoop*, *MapR*, *GridGain*, *Riak* und *MongoDB* sind bezüglich der Betriebssysteminteroperabilität voll kompatibel, einzig *Qizmt* ist nur auf der Windows-Plattformen lauffähig. Die Skalierung zur Laufzeit ist bei fast allen betrachteten Implementierungen möglich. *Qizmt* muss zum Hinzufügen oder Entfernen eines Knotens heruntergefahren werden, *MongoDB*-Cluster müssen zuerst durch Datenmigration auf Knotenentfernung vorbereitet werden.
Ein deutliches Unterscheidungsmerkmal stellt die Lastbalancierung dar: Dieses reicht bei den betrachteten Implementierungen von umfangreich und frei konfigurierbar bei *MapR* und *GridGain* bis nicht vorhanden bei *Qizmt*.
Einen guten Eindruck machte die Ausfallsicherheit der Cluster: Datenredundanz durch Replikation sowie die Unterstützung von verteilten Dateisystemen sind die Norm. *Single Points of Failure* sind zwar bei einigen Implementierungen vorhanden, die Auswirkungen dieser lassen sich jedoch meist über eine geeignete Clusterkonfiguration abmildern. Durch einen besonders großen Umfang an Konfigurationsmöglichkeiten und Sicherheitsfunktionen stachen hier *MapR* und *Riak* hervor.
Bezüglich der Fehlertoleranz verhalten sich die betrachteten Implementierungen insgesamt ähnlich: Im Falle eines Knotenausfalls werden die Tasks des ausgefallenen Knotens an einen neuen Knoten weitergereicht. Hier konnte sich *GridGain* positiv hervortun, da es als einzige der betrachteten Implementierung dazu in der Lage ist, nach einem kurzzeitigen Verbindungsabbruch

eines Knotens diesen mitsamt seiner bereits erbrachten Arbeitsleistung wieder in den Cluster aufzunehmen.
Die Monitoringmöglichkeiten der betrachteten Implementierungen waren insgesamt umfangreich. Mit Ausnahme von *Qizmt* konnte bei allen auf die erwarteten Metriken zugegriffen werden. Vom Umfang her konnten sich hier *Apache Hadoop* und *MapR* profilieren: *Vaidya* und *Hadoop Metrics* sind deutlich mächtiger, als die vergleichbaren Schnittstellen der anderen Implementierungen.
Die Qualität der *Dokumentation* der betrachteten Implementierungen unterscheidet sich stark. Während sich einige als sowohl umfang- als auch hilfreich erwiesen, wie dies zum Beispiel bei *MapR* der Fall ist, befanden sich vor allem die von *Hadoop* und *Qizmt* in einem desolaten Zustand.

5.3 Abschließende Worte

Während des Verlaufs der Fachstudie haben wir sowohl theoretisches, als auch praktisches Wissen über die MapReduce-Implementierungen sammeln können. Dabei stellten wir fest, dass sich die Implementierungen je nach Anwendungsbereich auch stark bezüglich des Schwierigkeitsgrades bei der Konfiguration, Auftrags-Erstellung oder der Bedienung unterscheiden. Zwar konnten wir aus Zeit- und Ressourcengründen die Implementierungen nur eingeschränkt betrachten, sowie nur sechs Kandidaten etwas näher untersuchen, haben aber dennoch Tendenzen festgestellt, so dass uns manche für den jeweiligen Gebrauch mehr oder weniger geeignet erscheinen.
So haben wir bei *Apache Hadoop* festgestellt, dass sich speziell die komplexe Konfiguration eher für Personen mit hohen Fachkenntnissen eignet. Deutlich einfacher und komfortabler arbeitet es sich mit den zahlreichen *Hadoop*-Distributionen wie zum Beispiel *MapR*, welches durch diverse integrierte Erweiterungen und der sinnvollen Benutzeroberfläche, sowie einer verständlichen Dokumentationen überzeugt hat. Dem produktiven Einsatz kommt außerdem der zusätzliche Support der Hersteller zugute. Läuft *Hadoop* oder eine seiner Distributionen einmal, so bekommt man ein mächtiges MapReduce-Tool, welches ansonsten so gut wie keine Wünsche offen lässt. Wir konnten zwar nur *MapR* näher betrachten, jedoch ist es sicherlich lohnenswert, sich auch die anderen Distributionen, wie zum Beispiel *Cloudera*, anzuschauen,.
GridGain überzeugte uns durch die problemlose Installation und die zahllosen Konfigurationsmöglichkeiten. Durch die Betriebssystemunabhängigkeit sowie die vielen unterstützten Programmiersprachen zeichnet sich diese Implementierung unter anderem durch hohe Flexibilität aus. In unseren Testfällen stießen wir auf keinerlei Probleme und der Entwicklersupport war vorbildlich. Die regelmäßig neu erscheinenden Versionen der Software und die zahlreichen Monitoringmöglichkeiten bieten weiterhin eine gute Grundlage für einen produktiven Einsatz.
Arbeitet man viel mit Datenbanken, so bekommt man mit *MongoDB* und Basho *Riak* zwei gute, MapReduce-fähige, NoSQL-Datenbanken mit jeweils unterschiedlichen Stärken und Schwächen. *MongoDB* lässt sich sehr einfach einrichten, das Schreiben und Ausführen der Aufträge geht schnell von der Hand – auch wenn MapReduce eigentlich nur Beiwerk der NoSQL-Datenbank ist. Die *Dokumentation* ist ausgezeichnet und führt gut durch alle nötigen Schritte. Dies ist bei *Riak* weniger der Fall, es sind etwas breitere und tiefere Kenntnisse, unter anderem mit Datenbanken, nötig - so arbeitet man beispielsweise viel über Konsolenbefehle und Konfigurationsdateien. Ist man jedoch erst einmal mit *Riak* vertraut, so bekommt man ein sehr mächtiges und stabiles System mit vielen Sicherheitsfeatures und Möglichkeiten, mit vielen unterstützen Sprachen und einer starken Datenbank. Bei der Funktions- und Featurevielfalt liegt *Riak* gegenüber *MongoDB* vorn.
Mit *Qizmt* lassen sich MapReduce-Aufträge zwar sehr einfach und schnell erstellen, die mangelhafte Dokumentation und der beinahe ausgestorbene Support lassen jedoch an positiven

Zukunftsaussichten der Implementierung berechtigte Zweifel aufkommen.
Für die Zukunft interessant erscheint *Stratosphere*. Auch wenn die Software noch im Anfangsstadium steckt und viele Funktionen noch nicht implementiert sind, so ist es durch seine neuen Ideen und das PACT-Modell als Generalisierung des MapReduce-Paradigmas sicherlich lohnenswert, dieses Projekt zu beobachten.

Literaturverzeichnis

[DNS06] Jeffrey Dean und Sanjay Ghemawat, MapReduce: Simplied Data Processing on Large Clusters, USENIX Association, pp. 137-150, 2006

[SPF] wikipedia.org, Single Point of Failure, 2012, http://de.wikipedia.org/wiki/Single_Point_of_Failure [Stand 26.06.2012]

[LBC] http://www.webopedia.com, What is load balancing?, 2012, http://www.webopedia.com/TERM/L/load_balancing.html [Stand 02.07.2012]

[HAD] The Apache Software Foundation, Welcome to Apache™ Hadoop™!, 2012, http://hadoop.apache.org/ [Stand 12.05.2012]

[MAR] MapR Technologies, Inc., MapR, 2012, http://www.mapr.com/ [Stand 26.06.2012]

[MAS] MapR Technologies, Inc., MapR Makes Hadoop Run Faster, 2012, http://www.mapr.com/products/why-mapr/fast [Stand 22.06.2012]

[CLD] Cloudera, Inc., Cloudera, 2012, http://www.cloudera.com/ [Stand 26.06.2012]

[CAS] Concurrent, Inc., Cascading, 2012, http://www.cascading.org/ [Stand 22.06.2012]

[RIA] Basho Technologies, Inc., Basho Riak Overview, 2012, http://basho.com/products/riak-overview/ [Stand 22.06.2012]

[AST] Teradata Corporation, Teradata Aster MapReduce Platform, 04.2012, http://www.asterdata.com/product/index.php [Stand 24.04.2012]

[MDB] 10gen, Inc., MongoDB, 2012, http://www.mongodb.org/ [Stand 26.06.2012]

[NOD] Nokia Corporation and the Disco Project, disco - massive data - minimal code, 2012, http://discoproject.org [Stand 02.07.2012]

[DIS] The Disco Project, Disco FAQ, 2012, http://discoproject.org/doc/disco/faq.html [Stand 26.04.2012]

[SPA] AMPLab, Spark - Lightning-Fast Cluster Computing, 2012, http://www.spark-project.org [Stand 23.07.2012]

[QIZ] Google Project Hosting, MySpace Qizmt - MySpace's Open Source Mapreduce Framework, 2012, http://code.google.com/p/qizmt [Stand 18.05.2012]

[QIT] Myspace LLC. , Qizmt on EC2 Tutorial, 2012, http://code.google.com/p/qizmt/wiki/MySpaceQizmtOnEC2Tutorial [Stand 26.06.2012]

[GRM] user: ronaldbradford, Getting Started with Gearman, 2010, http://gearman.org/index.php?id=getting_started [Stand 22.04.2012]

[SRA] speaker: Prof. Volker Markl, Stratosphere - Above the Clouds, 2012, https://stratosphere.eu/ [Stand 22.05.2012]

[ADV10] Alexandrov et al., Massively Parallel Data Analysis with PACTs on Nephele, 2010

[GRG] GridGain Systems, Inc., GridGain - Real Time Big Data, 2012, http://www.gridgain.com/ [Stand 22.05.2012]

[SEC] The Sector Alliance., Sector/Sphere, 2012, http://sector.sourceforge.net [Stand 26.05.2012]

[MRS] MapR Technologies, Inc., MapR Skalierung, 2012, http://www.mapr.com/products/only-with-mapr/scalable [Stand 03.07.2012]

[MRA] Amazon.com, Inc., Amazon Cloud mit MapR, 2012, http://aws.amazon.com/de/elasticmapreduce/mapr/ [Stand 03.07.2012]

[MRG] MapR Technologies, Inc., Google Cloud mit MapR, 2012, http://www.mapr.com/mapr-google [Stand 03.07.2012]

[GGF] GridGain, GridGain Features, 2012, http://www.gridgain.com/features [Stand 27.06.2012]

[RIR] Basho Technologies, Inc., Riak Clustering, 2012, http://wiki.basho.com/Concepts.html#Clustering [Stand 03.07.2012]

[TMS] Emma Barnett, MySpace loses 10 million users in a month, 2011, http://www.telegraph.co.uk/technology/myspace/8404510/MySpace-loses-10-million-users-in-a-month.html [Stand 26.06.2012]

Anhang

Anhang 1.1 Übersicht über mögliche Implementierungen

Anhang 1.2 Daten-Tabelle der Implementierungen

Anhang 1.3 KO-Tabelle für die Auswahl der Implementierungen

Anhang 2.1 Ergebnisse der Punktebewertung

Anhang 2.2 Überblick über die Bewertungsergebnisse

Anhang 2.3 Vorauswahl Bewertungsergebnisse im Detail

Anhang 3.1 Übersicht über die Ergebnisse der detaillierten Betrachtung

Anhang 1.1 Übersicht über Mögliche Implementierungen

MapReduce Implementations	Opensource License	Commercial License	Latest Release	Developement Activity	Supported Platforms
Hadoop-based distributions					
Apache Hadoop	Apache License 2.0	none	17.10.2011	Active – Nov. 2011	Independent – Java
HortonWorks	Apache License 2.0	commercial	Nov. 2011	Active	Independent – Java
MapR	?	free and commercial	Dez. 2011	Active	Independent – Java
Platform MapReduce	?	commercial	?	Active	Independent – Java
Cloudera	?	commercial	?	Active	Independent – Java
Greenplum MapReduce	?	Enterprise / community	?	Active	Independent – Java
Apache Hive	Apache License 2.0	none	16.12.2011	Active	Independent – Java
Apache Pig	Apache License 2.0	none	5.10.2011	Active	Independent – Java
Dumbo	Apache License 2.0	none			Independent – Java
Rhipe	Apache License 2.0	none	19.12.2011	Active	Independent – Java
Cascading	GNU GPL 3+	none	16.11.2011	Active	Independent – Java
Non-Hadoop implementations					
Basho Riak	Apache License 2.0	Riak Enterprise	17.11.2011	Active – Nov. 2011	Independent – Java & More
MongoDB	GNU AGPL 3.0	Silver / Gold	22.11.2011	Active – Nov. 2011	OSX, Win, Linux, Solaris
Aster Data Systems	none	Enterprise Edition	Demo	?	Windows, Linux
Nokia Disco	Nokia License	none	23.09.2011	Active – Nov. 2011	OSX, Linux
GPMR			none		Nvidia GPU based
Phoenix	BSD License	none	06.06.2011	Active, Jun. 2011	Linux, Solaris
Elastic Phoenix	A.W. Gordon License	none	Apr. 2011	Active – Apr. 2011	Linux
Spark	BSD License	none	14.07.2011	Active – Dec. 2011	Independent – Java
MPI-MR	modified BSD License	none	20.07.2011	Active – Sep. 2011	Linux
Filemap	New BSD License	none	14.01.2011	Active – March 2011	Unix
Plasma MapReduce	GNU GPL	none	Nov. 2011	Active – Oct. 2011	Linux
Mapredus	Mapredus License	none	05.08.2010	Active – Sep. 2010	Linux
MySpace Qizmt	GNU GPL 3.0	none	Dec. 2010	Active – Sep. 2011	Win Vista/7, 2k3,2k8 Sever
Microsoft Daytona	MSR-LA	none	28.11.2011	Active – Nov. 2011	Windows Azure
Misco			none		

HTTPMR	GNU GPL 3.0	none	none	Inactive	Linux
Skynet	MIT/X Consortium	none	31.05.2008	Inactive	Independent – Java
Starfish MapReduce	BSD License	none	30.03.2007	Inactive	Linux
Mincemeat	MIT License	none	2010	?	Needs Python Std Library
Mars	HKUST License	none	16.11.2009	Inactive	Linux + Nvidia GPU

MapReduce-ish implementations

Stratosphere	Apache License 2.0	none	18.08.2011	Active	Independent – Java
Gearman	BSD License	none	2011	Active solutions – 11.2011	Independent – Java & More
Sector/Sphere	Apache License 2.0	none	18.04.2011	Active – Nov. 2011	OSX, Windows, Linux
Galago's TupleFlow	New BSD License	none	May 2009	Active – Nov. 2011	Independent – Java
MapReduceTitan (=MPI-MR)	BSD License	none	20.07.2011	Active – Sep. 2011	Linux
Gridgain	GNU GPL 3.0	Gridgain Enterpris	08.11.2011	Active – Nov. 2011	OSX,Win,Linux,AmazonOS
LexisNexis HPCC	HPCC GNU AGPL 3.0	HPCC Enterprise E	27.10.2011	Active – Oct. 2011	Linux
Octopy	New BSD License	none	Apr. 2008	Inactive	OSX, Win, Linux
bashreduce	no license	none	23.03.2009	Inactive	independent
Microsoft Dryad	MSR-LA	none	17.11.2009	Stopped – Nov. 2011	Windows

Legende:		Aktiv
		Inaktiv

Anhang 1.2 Daten-Tabelle

MapReduce Implementations	Key Properties
Hadoop-based distributions	
Apache Hadoop	big, famous, classic MapReduce implementation of Google, has single point of failures
HortonWorks	
MapR	complete distribution around Hadoop and its add-ons
Platform MapReduce	
Cloudera	complete distribution around Hadoop and its add-ons
Greenplum MapReduce	
Apache Hive	- querying and managing large datasets residing in distributed storage - Built on top of Apache Hadoop - Query execution via MapReduce without the need to write java code
Apache Pig	- Simplicity. The parallel execution of complex analysis is easily understandable and workable. - Optimization. Pig optimized independently performing complex operations on the Carsten method. - Extensibility. Pig can be extended by custom functionality and thus individual Applications adapt. - Allows you to run nativeMapReduce functions (via .jar call)
Dumbo	Dumbo is a project that allows you to easily write and run Hadoop programs in Python - Dumbo is a Hadoop library for python
Rhipe	Divide & Recombine for the Analysis of Large Complex Data. RHIPE is the merger of the R ILDA for data analysis, and the Hadoop the distributed file system (HDFS) and compute engine (MapReduce) first invented by Google and now supported by the Apache Software Foundation.
Cascading	Cascading is a thin Java library that sits on top of Hadoop's MapReduce layer. It is not a new text based query syntax (like Pig) or another complex system that must be installed on a cluster and maintained (like Hive). Though Cascading is both complementary to and is a valid alternative to either application.
Non-Hadoop implementations	
Basho Riak	Buckets/keys = Riak objects, which can store links and meta data exposed as HTTP-headers and arrays in MapReduce-jobs/can do ALOT more
MongoDB	Similiar to Hadoop, SQL-group-by-like
Aster Data Systems	SQL-MapReduce-Framework, in database analytics, fraud detection, graph analysis, sharing behavior, Hadoop integration, uses MPP, supports Cloud-systems
Nokia Disco	Distributed computing framework based on MapReduce, own Disco Distributed filesystem (DDFS) for storing and processing of petabytes of log data/photos/videos/discodb indices, has fault-tolerance and high-availability through metadata replication
GPMR	GPU-MapReduce single library using C++ and CUDA, speedy within nodes, not so speedy across nodes, claims: faster than Phoenix and Mars on single nodes
Phoenix	Shared-memory implementation of Google's MapReduce for improved multicore-usage within each node

Elastic Phoenix	Phoenix based, but in comparison can add more clients to an already started MapReduce job, for this: at least 1 master (process splitting and assigning jobs) and 1 worker process (executing jobs), can create Vms for more flexibility while splitting jobs
Spark	Data analytics, focus on in-memory-cluster-computing for speed-up, 30x faster than Hadoop in iterative algorithms for machine learning and interactive data-mining, can do general data processing and coexist with Hadoop on Apache Mesos clusters.
MPI-MR	Based on MPI massage passing and OINK, small, no fault tolerance, MapReduce library in C++
Filemap	File- not tuple-based, automatic caching and reuse of results, data replication, no master node, no installation required
Plasma MapReduce	Distributed file system, key/value db, MapReduce system, performance oriented = uses more shared-memory for less network traffic, PlasmaFS = own filesystem
Mapredus	MapReduce Framework using redis (key value storage server) and resque (Ruby library for creating backround jobs on multiple queues)
MySpace Qizmt	DataMining, Analytics, Bulk Media Processing, Content Indexing
Microsoft Daytona	Iterative MapReduce runtime, data analytics, machine learning algortihms, designed for clouds
Misco	Distributed computing framework for mobile devices, master server commands and worker clients process and return results
HTTPMR	MapReduce implementation of Google for HTTP-server-clusters: access, requests, processing of nodes and clusters via HTTP
Skynet	Ruby, Adaptive, self-upgrading, fault-tolerant, fully distributed system, no single point of failure, no master servers, workers can act as masters
Starfish MapReduce	Distributed Programming
Mincemeat	Extremely lightweight single python-file with <13kb, workers = clients can join and leave the cluster at any time during processing, code authentication generated for both ends for security
Mars	Claims: up to 16x faster than Phoenix, which shall be the state of the art MapReduce-implementation, uses the GPU

MapReduce-ish implementations

Stratosphere	While Nephele/PACT takes away much work from the programmer, the actual process of denying a given problem , PACT program still has to be done manually. The automatic generation of PACT programs from declarative languages like SQL or XQuery is a field of research. The PACT programming model is a generalization of the well-known MapReduce programming model, extending it with further second-order functions, as well as with Output Contracts that give guarantees about the behavior of a function.
Gearman	Multilanguage, lightweighted, no single point of failures. Client, worker, job-server: client creates job, sends to job server, job server sends to worker, worker reports back to the client through the job server
Sector/Sphere	Distributed data storage and processing, secure distributed file system, claims: constant 2-4x faster than Hadoop, WAN support, Software Level Fault Tolerance – less dependant on RAIDs, data management rules changeable during runtime
Galago's TupleFlow	TupeFlow: Mix of MapReduce, make/ant, database. Structured Retrieval, DumpIndex, JobExecutionContext
MapReduceTitan (=MPI-MR)	Like Google's MapReduce, but reduction is to a single value, uses a MPI-MapReduce-Library which takes function pointers and operates on Key-Value and Key-Multiple-Value, # of map calls = # of MPI processes

Gridgain	"real time streaming" MapReduce for example Online-video-Feeds, distributed functional framework for Java, two elements: grid task (task description, MapReduce logic) and grid job (defines work units that go to remote nodes, result comes back to the task and gest reduced to final result)
LexisNexis HPCC	claim: faster than Hadoop, has an IDE with its own programming language ECL optimized for parallel directives, uses two clusters, one for parallel batch-data processing (=Thor) and one for online query applications using indexed data files (= Roxie), each of them can be parallelized within themselves too
Octopy	Fusion of Starfish and Google's MapReduce
bashreduce	Single bash-script-Hadoop-replacement with simplified features.
Microsoft Dryad	Infrastructure for small clusters to large data-centers. Is a directed acyclic graph generator, programs = vertices, channels which direct programs = edges, both can change during execution

Legende:

Aktiv	
Inaktiv	

Anhang 1.3 KO-Tabelle

MapReduce Implementations	Popularity	Documentation / Coverage / Community	Installation	Unique Features
Hadoop-based distributions				
Apache Hadoop	very high	available / very high / very high	Installation guide	N/A
HortonWorks	low	available / very low / very low	none	N/A
MapR	high	available / high / high	Installation guide	MapR NFS
Platform MapReduce	very low	available / mediocre / very low	none	N/A
Cloudera	high	available / high / very high	Installation guide	N/A
Greenplum MapReduce	very low	available / very low / mediocre	Installation guide	N/A
Apache Hive	mediocre	available / high / mediocre	Installation guide	defines a simplequery languagecalled QL, that enables users query the data
Apache Pig	mediocre	available / mediocre / mediocre	Installation guide	consists of a high-level language for expressing data analysis programs
Dumbo	very low	available / low / low	Installation guide	allows you to write and run Hadoop programs in Python
Rhipe	very low	available / very low / very low	Installation instructions	merger of the R ILDA for data analysis, HDFS and MapReduce
Cascading	low	available / high / high	Installation guide	thin Java library that sits on top of Hadoop's MapReduce layer
Non-Hadoop implementations				
Basho Riak	high (1)	available / middle / high	available	written in Erlang
MongoDB	very high	available / high / very high	available	N/A
Aster Data Systems	high (1, 2)	available / very low / low	none	N/A
Nokia Disco	mediocre	available / high / mediocre	available	N/A
GPMR	none	available / very low / very low	none	N/A
Phoenix	low	available / very low / low	none	N/A
Elastic Phoenix	low	available / low / low	available (1)	Tested ONLY under Linux on 64-bit x86 and running Elastic Phoenix in virtual machines requires the Nahanni shared memory system, which is available ONLY for Linux KVM
Spark	low	available / mediocre / low	available	N/A
MPI-MR	low	available (1) / high / mediocre	available	N/A
Filemap	none	available / very low / very low	available	N/A
Plasma MapReduce	low	available / mediocre / low	available (1, 2)	N/A
Mapredus	none	available / none / none	available	N/A
MySpace Qizmt	mediocre (1, 2)	available / high / mediocre	avalable (1, 2)	own IDE for MR-Jobs
Microsoft Daytona	low	available / high / none	available	azure clouds only
Misco	none	none / none / none	none	N/A
HTTPMR	none	none / none / none	none	N/A
Skynet	low	available (1) / high / low	available (1)	N/A

Starfish MapReduce	very low	available (1) / mediocre / very low	available	N/A
Mincemeat	none	none / none / none	none	N/A
Mars	low	available / very low / none	none	N/A

MapReduce-ish implementations

Stratosphere	very low	available / mediocre / low	Installation guide	uses the the PACT Programming Model
Gearman	Significant search volume as of 03/10	Doc+Tutorials / mediocre / Not centralized	Installation guide/Tarball	N/A
Sector/Sphere	Sporadic use in research environments	Documentation / mediocre / Live forum	Installation guide/Tarball	N/A
Galago's TupleFlow	Purely educational	Short guide / low / No community	No source	N/A
MapReduceTitan (=MPI-MR)	Obscure	Wiki / low / No MR community	Installation guide/executable	Uses MPI-MR
Gridgain	Appears in press sporadically	Manual / mediocre / Support	Installation guide/executable	N/A
LexisNexis HPCC	LexisNexis itself is fairly well known	No data	No source	only runs on proprietary clouds
Octopy	Obscure	Wiki / very low / none	User guide/Python script	Python
bashreduce	Obscure	none / none / none	Batchfile	Batch script
Microsoft Dryad	Discontinued before release	none / none / none	N/A	N/A

Legende:	Aktiv
	Inaktiv
	Aussortiert
	KO-Kriterium

Einschätzung der Relevanz, von sehr niedrig bis sehr hoch:

very low < *low* < *mediocre* < *high* < *very high*

Anhang 2.1 Ergebnisse der Punktebewertung

Im Folgenden können alle Implementierungen der Vorauswahl, ihre Teilpunkte und das Gesamtergebnis in einer Tabelle begutachtet werden:

Implementierung	*Map-Reduce (max. 10 Pkt)*	*Datenbank / Dateisystem (max. 7 Pkt)*	*Dokumentation (max. 5 Pkt)*	*Installation (max. 5 Pkt)*	*Besonder-heiten (max. 5 Pkt)*	*Gesamt (max. 32 Pkt)*
Apache Hadoop	3	0	3	1	3	**10**
MapR	6	5	4	2	2	**19**
Cloudera	6	3	1	2	2	**14**
Cascading	3	0	1	0	1	**5**
Basho Riak	5	5	4	2	2	**18**
Aster Data SQL-MapReduce	6	2	0	0	2	**10**
MongoDB	2	2	3	2	1	**10**
Nokia Disco	4	2	1	0	1	**8**
Spark	5	4	-1	0	3	**11**
MySpace Qizmt	6	3	2	1	1	**13**
Gearman	3	-2	3	0	3	**7**
Stratosphere	4	1	2	1	4	**12**
Gridgain	4	1	4	2	4	**15**
Sector/Sphere	3	3	0	-1	2	**7**

Anhang 2.2 Überblick über die Bewertungsergebnisse

	Apache Hadoop	MapR	Cloudera	Cascading	Basho Riak	Aster Data SQL-MapReduce	Mongo DB	Nokia Disco	Spark	MySpace Qizmt	Gearman	Stratos phere	Gridgain	Sector/ Sphere
MapReduce-Implementierung														
Vorhandensein	0	0	0	0	0	0	0	0	-	0	-	-	0	0
Flexibilität	+	+	+	+	+	+	-	0	+	0	+	0	+	-
Skalierbarkeit	0	+	+	0	0	+	0	+	0	+	0	0	0	0
Einfachheit	+	+	0	+	+	+	0	-	+	+	0	+	+	0
Boni (max 6)	+	+++	++++	+	+++	+++	+++	++++	++++	++++	+++	++++	+++	++++
Datenbank/Dateisystem														
Vorhandensein	0	0	0	0	0	0	-	0	0	0	-	0	-	0
Skalierbarkeit	0	0	0	0	+	0	0	0	+	0	0	0	0	0
Sicherheit	0	+	+	0	+	-	+	0	+	+	-	-	-	+
Einfachheit	-	+	+	-	+	+	+	+	0	+	0	0	+	0
Boni (max 3)	+	+++	+	+	++	+	+	+	++	+		++	++	++
Dokumentation														
Vorhandensein	0	0	0	0	0	*	0	0	0	0	0	0	0	0
Ausführlichkeit	+	+	+	+	+		+	0	0	0	+	0	+	-
Komfort	0	0	-	-	0		0	0	-	+	0	+	+	0
Boni (max 2)	++	+++	+	+	+++		++	+		+	++	+	++	+
Installation														
Installationsanleitung	0	0	+	0	0		0	0	0	0	0	0	0	0
Betriebssyteme	0	+	0	0	+		+	0	0	0	-	0	+	-
Boni (max 1)	+	+	+		+		+			+	+	+	+	0
Besonderheiten	+++	++	++	+	++	++	+	+	+++	+	+++	++++	++++	++
Endergebnis	+10	+19	+14	+5	+18	+10	+10	+8	+11	+13	+7	+12	+15	+7

* Konnte nicht überprüft werden

Anhang 2.3 Vorauswahl Bewertungsergebnisse im Detail

Hadoop-basierende MapReduce-Implementierungen

Apache Hadoop

MapReduce-Implementierung (+3)

0 Google's MapReduce
+ Flexibilität durch Erweiterungen wie Apache Hive und Pig
0 Skaliert auf tausende Knoten (mit mehr Geschwindigkeit laut Apache [HAD])
+ Nutzung kann durch Erweiterungen wie Hive QL / Pig Latin vereinfacht werden

---

+ Analysemöglichkeiten über Hive/Pig

Datenbank und Dateisystem (+0)

0 HDFS / HBase
0 Skalierung des Systems durch mehr Knoten
0 Fehlertoleranz
- Einrichtung eines Hadoop Clusters über Konsolenbefehle

+ Durch Apache Mesos kann Hadoop Cluster-Daten mit anderen System teilen

Dokumentation (+3)

0 Online-Dokumentation
+ sehr ausführlich sowohl für MapReduce, als auch für HDFS inkl. Guides
0 komfortabel durch Inhaltsangabe, Direktlinks und Querverweise

---

+ Tutorials
+ Weltweite Nutzerbasis mit Wiki, Foren und Benutzerseiten

Installation (+1)

0 Installations- und Konfigurationsanleitungen
0 Linux

+ fertige Binaries

Besonderheiten (+3)

Bewertungsergebnis: **+10**

MapR

MapReduce-Implementierung (+6)

0 Google's MapReduce
+ Flexibilität durch Erweiterungen wie Apache Hive und Pig
+ Skaliert auf tausende Knoten (mit überproportional mehr Geschwindigkeit als Hadoop laut MapR [MRS])
+ vereinfachte Handhabung / Erstellung von MapReduce-Aufträgen, Apache Hive QL / Pig Latin

---

+ Analysemöglichkeiten über Hive/Pig
+ MapR Control System
+ dezentrale Auftragsüberwachung

Datenbank und Dateisystem (+5)

0 HDFS / HBase
0 Skalierung des Systems durch beliebig viele Knoten
+ Fehlertoleranz, automatische Spiegelungen/Snapshots, Datenrechte und Quotas
+ Einrichtung und Konfiguration über Installer, MapR Heat und MCS

+ MapR NFS
+ Überwachung und Management über MapR Heat und MCS
+ Weitere Schnittstellen über Apache Mesos

Dokumentation (+4)

0 Online Dokumentation
+ sehr ausführlich und detailliert mit Beispielen
0 Komfortabel durch Inhaltsangabe, Querverweise

+ aufwendig Videos und Demos
+ Forum, Blog, Training
+ direkter Entwicklersupport

Installation (+2)

0 Installations- und Konfigurationsanleitungen
+ Linux, Windows, Mac

+ Installer

Besonderheiten (+2)

Bewertungsergebnis: **+19**

Cloudera

MapReduce-Implementierung (+6)

0 Google's MapReduce
+ Durch die zusätzlichen Pakete viele Möglichkeiten der Mapreduce-Implementierung
+ Skalierbar, höhere Performance bei schwierigen Aufgaben [CLD]
0 Einfache Handhabung und Kontrolle der Aufträge dank der vielen Distributions-integrierten Tools

---

+ Bei Ausfall werden ausstehende MapReduce-Aufträge gesichert
+ Viele Analyse- und Monitoring-Tools
+ GUIs für viele Tools
+ Erweiterbarkeit der Funktionalität durch Schnittstellen

Datenbank und Dateisystem (+3)

0 Apache Hadoop's HDFS integriertes Dateisystem
0 Skalierbar [CLD]
+ Bei Ausfall können Master/Worker-Knoten ersetzt werden
+ Tools/GUI zum Konfigurieren der Cloud

+ Management-Tools zur Überwachung der Cloud

Dokumentation (+1)

0 Online Dokumentation
+ Ausführlich für jedes Modul
- Teilweise sehr umständlich, da für jedes Paket eigene Dokumentation vorhanden

---

+ aktive Nutzerbasis (auch Hadoop und Module) sowie zahlreiche Mailing-Lists

Installation (+2)
+ Installations- und Konfigurationsanleitung vorhanden
0 Linux

+ Ausführbare Binärdatei

Besonderheiten (+2)

Bewertungsergebnis: **+14**

Cascading

MapReduce-Implementierung (+3)
0 Google's MapReduce
+ Flexibilität durch Möglichkeit der Implementierung von MapReduce-Aufträgen in veschiedenen Skriptsprachen
0 Skaliert da Apache Hadoop basierend
+ Durch die Verwendung von gekapselten Prozessen wird Handhabung der Aufträge vereinfacht

---

+ Analysemöglichkeiten über Hadoop Zusatzpakete

Datenbank und Dateisystem (+0)
0 HDFS / HBase
0 Skalierung des Systems durch mehr Knoten [CAS]
0 Fehlertoleranz
- Einrichtung eines Hadoop-Clusters über Konsolenbefehle

+ Durch Apache Mesos kann Hadoop Cluster-Daten mit anderen System teilen

Dokumentation (+1)
0 Online-Dokumentation
+ sehr ausführlich, auch API-Dokumente werden zu Verfügung gestellt
- fehlende Komfortfunktionen wie Stichwortsuche

+ Nutzerbasis-Webseite mit Neuigkeiten und aktuellen Informationen sowie Mailing-List

Installation (+0)
0 Installations- und Konfigurationsanleitung nur Textdatei
0 Linux

Besonderheiten (+1)

Bewertungsergebnis: **+5**

Nicht auf Hadoop basierende Implementierungen von MapReduce

Basho Riak

MapReduce-Implementierung (+5)

0 Riak Pipe
+ Flexibilität durch Erlang und Javascript
0 Skaliert mit weiteren Knoten mit mehr Geschwindigkeit [RIR]
+ vereinfachte Steuerung durch HTTP API möglich

+ Large-Scale-Analyse
+ Riak Control
+ kein Single-Point-of-Failure

Datenbank und Dateisystem (+5)

0 eigenes Datenbanksystem
+ skaliert während des Betriebs mit weiteren Rechnern
+ Replikation, Fehlertoleranz, Wiederherstellungsoptionen
+ Einrichtung und Konfiguration über Installer und Riak Control vereinfacht

---

+ Überwachung und Management über Riak Control
+ Weitere Schnittstellen über HTTP zu eigenen und fremden Tools

Dokumentation (+4)

0 Dokumentation online und im Downloadpaket
+ sehr ausführlich und detailliert mit Beispielen
0 Komfortabel durch Inhaltsangabe, Querverweise

+ Video zur Nutzung der GUI, inoffizielle Videos
+ Mailing-List, Nutzerbasis-Webseiten, Blog, Facebook, Twitter, Real-Live-Meetings
+ direkter Entwicklersupport

Installation (+2)

0 Installations- und Konfigurationsanleitungen
+ Linux, Mac OS

+ Installer

Besonderheiten (+2)

Bewertungsergebnis: **+18**

Aster Data SQL-MapReduce

MapReduce-Implementierung (+6)

0 MapReduce-gleiche Implementierung
+ Möglichkeit, den Cluster selbst zu erweitern
+ Einfache Erstellung von MapReduce-Aufträgen
+ Unterstützung vieler Programmiersprachen

+ Tools für das Konfigieren der Cloud
+ Viele Analyse- und Monitoring-Tools
+ GUI für Montitoring/Lastbalancierung

Datenbank und Dateisystem (+2)

0 Mitgelieferte Architektur für Datenbankcluster
0 skaliert [AST]
- Keine Aussage über Verhalten bei Hardwareausfall
+ Einfache Konfiguration über GUI

+ Diverse Management-Tools
+ Unterstützung gängiger SQL-Interfaces

Dokumentation/Installation (0)

0 Ohne eine kommerziell erworbene Version kann für beide Kriterien objektiv betrachtet keine Bwertung gegeben werden.

Besonderheiten (+2)

Bewertungsergebnis: **+10**

MongoDB

MapReduce-Implementierung (+2)

0 MapReduce-ähnliche Implementierung
0 Erstellung von MapReduce-Aufträgen unkompliziert möglich
- Keine weiteren Implementierungssprachen unterstützt

+ Kein Single Point of Failure, sichere Auftragsausführung durch Queuing
+ Viele Analyse- und Monitoring-Tools
+ GUI für Montitoring/Lastbalancierung

Datenbank und Dateisystem (+2)

- Kein eigenes verteiltes Dateisystem
+ Möglichkeit der Datensicherheit durch redundante Speicherung
+ Unkomplizierte Einrichtung

+ Sehr flexible Datenstruktur

Dokumentation (+3)

0 Dokumentation vorhanden
+ Ausführliche, verständliche Bedienungsanleitung und häufige Referenzierung
0 Gute Gliederung

+ Tutorials
+ Aktive Nutzerbasis

Installation (+2)

0 Installations- und Konfigurationsanleitungen
+Binärdistributionen für die gängigen Betriebssysteme (Win, Linux, Mac OS)

+ Kompilierungsleitfaden

Besonderheiten (+1)

Bewertungsergebnis: **+10**

Nokia Disco

MapReduce-Implementierung (+4)

0 Volle MapReduce Implementierung
0 Zwei verschiedene Sprachen für Auftragserstellung
+ Zur Laufzeit skalierbar [NOD]
- Wenig Monitoring-Möglichkeiten

+ Funktionalität zur Erhaltung von Datenlokalität
+ Flexibilität durch Python
+ Mehrere Worker auf einem Knoten möglich
+ Steuerung des Clusters über ein Webinterface

Datenbank und Dateisystem (+2)

0 Eigenes verteiltes Dateisystem
0 Skalierbar [NOD]
+ Eigenes Rapid-Access Indexierungsformat

+ Ausfallsicherheit durch Datenreplikation

Dokumentation (+1)

0 Online-Dokumentation
0 Adäquate Dokumentation
0 Verzeichnis

+ Tutorials

Installation (0)

0 Linux, MacOS, aber nur Quellcode/Debian-Pakete
0 Installationsanleitung vorhanden

Besonderheiten (+1)

Bewertungsergebnis: **+8**

Spark

MapReduce-Implementierung (+5)

- Kein Google MapReduce
0 Skalierbar [SPA]
+ Einfache Erstellung von MapReduce-Aufträgen dank Scala
+ automatische Arbeitslastumverteilung durch den Masterknoten

---

+ Analysetools
+ Verwendet eigene MapReduce-Implementierung
+ Hohe Fehlertoleranz und Auffangen der Aufträge
+ Schneller Zugriff auf Auftragsdaten dank Verwendung des Speichers und Prozessorcaches

Datenbank und Dateisystem (+4)

0 Nutzt Apache Mesos um auf ein Hadoop Cluster zugreifen zu können
+ sehr hoch skalierbar [SPA]
+ Hohe Fehlertoleranz bei Hardwareausfall
0 Abhängigkeit von Mesos bringt weitere Bedienkomponente ins Spiel

---

+ Webbasierte Monitoring-Tools
+ Bietet Schnittstellen für weitere Sprachen an

Dokumentation (-1)
0 Online-Dokumentation
0 Grundlegende Informationen vorhanden, viele Beispiele
- Zusätzliche Ressourcen sehr rar

Installation (+0)
0 Installations- und Konfigurationsanleitung vorhanden
0 Linux

Besonderheiten (+3)

Bewertungsergebnis: **+11**

Myspace Qizmt

MapReduce-Implementierung (+6)
0 Natives MapReduce
0 Skalierbar [QIZ]
+ Flexibilität durch mehrere MapReduce Modi
+ Erstellung von MapReduce-Aufträgen unkompliziert möglich
+ Zahlreiche Analysetools
+ Debugger für Aufträge
+ Eigene Entwicklungsumgebung
+ Dezentrale Steuerung

Datenbank und Dateisystem (+3)
0 Eigenes dediziertes Dateisystem
0 Skalierbarkeit [QIZ]
+ Sicherheit durch Datenredundanz
+ Konfigurationstools
+ Analystetool

Dokumentation (+2)
0 Online-Bedienungsanleitung vorhanden
0 Grundlegende Informationen enthalten
+ Beispiele, Bilder und Troubleshooting Guide
+ Tutorials, Wiki, Tracker

Installation (+1)
0 Installations- und Konfigurationsanleitung
0 Windows
+ fertige Binaries

Besonderheiten (+1)

Bewertungsergebnis: **+13**

Implementierungen mit MapReduce-ähnlichen Programmiermodellen

Gearman

MapReduce-Implementierung (+3)

- Keine native MapReduce-Funktionalität
0 Leicht skalierbar [GRM]
+ Sehr flexibel durch quasi unbegrenzte Interoperabilität [GRM], viele unterstützte Programmiersprachen
0 Erstellung von MapReduce-Aufträgen unkompliziert möglich

+ Einziger offensichtlicher SpoF lässt sich leicht verhindern (mehrere Auftragsserver), sichere Auftragsausführung durch Queuing
+ Master/Worker-Architektur für MapReduce-ähnliche Aufgaben geeignet
+ Analysefunktionalität vorhanden

Datenbank und Dateisystem (-2)

- Kein eigenes Dateisystem
- Kein Verthalten bei Hardwareausfall
0 Leicht skalierbar [GRM]

Dokumentation (+3)

0 vorhanden
+ Ausführliche, verständliche Bedienungsanleitung, häufige Referenzierung weiterführender Themen
0 Gute Gliederung

+ Tutorials
+ Aktive Nutzerbasis

Installation (+0)

0 Linux
- Installationsanleitung etwas unverständlich und dürftig

+ Durch verschiedene Auftragsserver-Varianten, aufgrund von Skriptsprachen weitestgehend Plattformunabhängig

Besonderheiten (+3)

Bewertungsergebnis: **+7**

Stratosphere

MapReduce-Implementierung (+4)

- Kein traditionelles MapReduce
0 Cloud-Modus wird noch nicht vollständig unterstützt
0 Einfache Erstellung von Map-Reduce(-ähnlichen) Aufträgen
+ Neue Aufgaben können per Interface direkt an den JobManager übertragen werden

---

+ Nephele Visualisation Tool zum Überwachen der Aufgaben
+ PACT-Modell als Generalisierung von MapReduce
+ Rückmeldung über Abstürze in einer Log-Datei
+ GUI für Überwachung und Auftrags-Übertragung

Datenbank und Dateisystem (+1)

0 Apache Hadoop's HDFS
0 Skalierbarkeit [SRA]
- Momentan noch keine Fehlertoleranz
0 Einfaches Konfigurieren des Clusters

+ Durch HDFS viele Tools zur Überwachung des Clusters installierbar
+ Aufgrund vieler Schnittstellen erweiterbar

Dokumentation (+2)

0 Online-Dokumentation
0 ausreichend ausführlich, einige Beispiele
+ Komfortable Bedienung, Suche, Querverweise, FAQs

+ Ticketsystem, Mailinglist

Installation (+1)

0 Installations- und Konfigurationsanleitung vorhanden
0 Linux

+ ausführbare Binaries sowie ausführbare Beispieldateien

Besonderheiten (+4)

Bewertungsergebnis: **+12**

Gridgain

MapReduce-Implementierung (+4)

0 Eigene MapReduce-ähnliche Implementierung
0 Skalierbar [GRG]
+ Einfache Erstellung von MapReduce-Aufträgen, mehrere Implementierungssprachen unterstützt
+ Möglichkeit zur Änderung der MapReduce-Aufträge in Echtzeit

---

+ Tools für das Konfigurieren der Cloud
+ Viele Analyse- und Monitoring-Tools
(+ GUI fürs Monitoring - nur Enterprise Edition)

Datenbank und Dateisystem (+1)

- Kein integriertes verteiltes Dateisystem
0 Skalierbar [GRG]
- Keine Angaben zu Verhalten bei Hardwareausfall
+ Tools/GUI zum Konfigurieren der Cloud

+ Management Tools zur Überwachung der Cloud
+ Angebot diverser Server-Provider-Interfaces

Dokumentation (+4)

0 Online-Dokumentation
+ Sehr ausführlich, viele Beispiele
+ Komfortable Bedienung, Suche, Querverweise, FAQs

+ Einige Videotutorials und Links
+ aktive Nutzerbasis und Forum

Installation (+2)
0 Installations- und Konfigurationsanleitung vorhanden
+ Windows, Linux, Mac OS

+ Nur entpacken der ZIP-Datei nötig

Besonderheiten (+4)

Bewertungsergebnis: **+15**

Sector/Sphere

MapReduce-Implementierung (+3)
0 Eigenes MapReduce
- Relativ komplizierte Bedienung und Erstellung von Aufträgen
0 Frei konfigurierbarer Grad an Parallelität
0 Mehrere Programmiersprachen verwendbar

+ Kein Single Point of Failure
+ Viele Analyse- und Monitoring-Tools
+ GUI für Montitoring/Lastbalancierung
+ Hoher Grad an Verarbeitungseffizienz

Datenbank und Dateisystem (+3)
0 Eigenes Dateisystem Sector
0 Skalierbar [SEC]
0 Einrichten der Cloud schwer einzuschätzen
+ Integrierte Datensicherheit

+ Hohe Lesegeschwindigkeit
+ Unkomplizierte Schnittstelle zu Sphere

Dokumentation (0)
0 Dokumentation vorhanden
- nicht sehr ausführlich

+ Hilfreiche Nutzerbasis

Installation (-1)
0 Theoretisch von allen OS unterstützt, nur Linux getestet
- keine Installationsanleitung
0 Quelltext öffentlich zugänglich

Besonderheiten (+2)

Bewertungsergebnis: **+7**

Anhang 3.1 Übersicht über die Detaillierte Betrachtung

Testkategorien	Apache Hadoop	Gridgain	MapR
MapReduce			
1. Erstellung			
1.1 Wortzahlermittlung	erfolgreich	erfolgreich	erfolgreich
1.3 Unterstützte Sprachen	Java (mit Streaming beliebige)	Java, Scala, Groovy	Java (mit Streaming beliebige)
1.4 Möglichkeiten zur Job-Erstellung/Änderung	Konsole/IDE-Unterstützung	per IDE	eigene Programmierung/Konsole
2. Ausführung			
2.1 Nötige Vorarbeit	mittel	niedrig-mittel	sehr niedrig
2.2 Job Queue API	vorhanden	vorhanden	vorhanden
2.3 Weiterverwenden der Ergebnisse	möglich	möglich	möglich
2.4 Kompatibilität bei verschiedenen OS	kompatibel, nicht empfohlen	voll kompatibel **	Clients für OSX und Windows
3. Skalierung			
3.1 Knoten hinzufügen	manuelle Konfiguration	möglich	möglich
3.2 Knoten hinzufügen zur Laufzeit	möglich	möglich	möglich
3.3 Knoten entfernen	manuelle Konfiguration	möglich	möglich
3.4 Knoten entfernen zur Laufzeit	möglich	möglich	möglich
3.5 Skaliebarkeit auf hohe Anzahl von Knoten	Bis 2000 in Praxiseinsatz	möglich **	mehr als 10.000 **
3.6 Automatische Skalierbarkeit	nein	möglich	möglich
4. Lastbalancierung			
4.1 Vorhandensein	nur per Hbase	vorhanden	vorhanden
4.2 Konfigurationsmöglichkeiten	vorhanden	umfangreich	vorhanden
Infrastruktur			
5. Dateisystem			
5.1 Import von Daten	Konsolenbefehl	möglich	möglich – sehr einfach
5.2 Export von Daten	Konsolenbefehl	möglich	möglich – sehr einfach
5.3 Unterstütze Dateisysteme	HDFS, S3, nativ über FTP/HTTP	nativ, erweiterbar	HDFS, Amazon&Google-Cloud
5.4 Zugriffsmöglichkeiten	eigene API	Sprachenabhängig	MapR – NFS (Direct Access)
5.5 Datenkonsistenz/Verhalten bei Ausfall	automatische Datenreplikation	Vorhanden, Segmentierungs-Check u. Mehr	automatische Datenreplikation, manuelle Snapshots und Clusterspiegelungen
5.6 Vorteile einer größeren Knotenanzahl	mehr Speicherplatz	Leistungssteigerung	mehr Replikationen möglich
6. Fehlertoleranz			
6.1 Ausfall eines Knotens	Tasks werden neu zugewiesen	Jobs/Daten werden weitergereicht	Tasks werden neu zugewiesen
6.2 Konfigurationsmöglichkeiten		umfangreich	vorhanden
6.3 Kurzeitiger Verbindungsverlust	Tasks werden neu zugewiesen	Diverse Möglichkeiten zur Synchronisierung	Tasks werden neu zugewiesen
7. Monitoring			
7.1 Aktive MapReduce-Aufträge	werden angezeigt	werden angezeigt	werden in der GUI angezeigt
7.2 MapReduce-Aufträge in der Queue	werden angezeigt	werden angezeigt	werden in der GUI angezeigt
7.3 Verfügbarkeit von Knoten	werden angezeigt	Anzeige möglich	wird in der GUI angezeigt
7.4 Arbeitslast von Knoten	werden angezeigt	wird angezeigt	wird in der GUI angezeigt
7.5 Aggregierte Sichten	Benutzerdefinierte Metriken, nur als Rohdaten	vorhanden	vorhanden
7.6 Logdateien	vorhanden – umfangreich	vorhanden	vorhanden – umfangreich
7.7 Oberflächen/GUI	WebUI für Basisdaten	KonsolenUI / GUI	MapR System Control
7.8 Monitoringmöglichkeiten	Basisdaten, bzw. per Drittsoftware	umfangreich	vorhanden – umfangreich
** laut Herstellerangaben			

Übersicht über die Detaillierte Betrachtung Teil 2

Testkategorien	Basho Riak	MongoDB	MySpace Qizmt
MapReduce			
1. Erstellung			
1.1 Wortzahlermittlung	erfolgreich	erfolgreich	erfolgreich
1.3 Unterstützte Sprachen	Erlang, JS, C/++, Java, Python	JavaScript	C#
1.4 Möglichkeiten zur Job-Erstellung/Änderung	Konsole, eigene Funktionen	Konsole/Kommando via Treiber	Konsole/eigener Editor
2. Ausführung			
2.1 Nötige Vorarbeit	hoch	keine	niedrig
2.2 Job Queue API	nicht vorhanden (nur mit Erweiterungen)	einzelner Queue pro Knoten	nicht vorhanden
2.3 Weiterverwenden der Ergebnisse	möglich	möglich	möglich
2.4 Kompatibilität bei verschiedenen OS	kompatibel	Div. OS kompatibel, nicht x64/x86	nicht kompatibel
3. Skalierung			
3.1 Knoten hinzufügen	einfach möglich	per Konsolenbefehl	per Konsolenbefehl
3.2 Knoten hinzufügen zur Laufzeit	möglich	Nur zur Laufzeit möglich	nicht möglich
3.3 Knoten entfernen	einfach möglich	Nur nach vorheriger Datenmigration	per Konsolenbefehl
3.4 Knoten entfernen zur Laufzeit	möglich	Nur zur Laufzeit möglich	nicht möglich
3.5 Skaliebarkeit auf hohe Anzahl von Knoten	möglich, keine Maxanzahl angegeben	Bis 100 getestet	möglich, keine Maxanzahl angegeben
3.6 Automatische Skalierbarkeit	möglich	möglich	möglich
4. Lastbalancierung			
4.1 Vorhandensein	vorhanden	Nicht dynamisch, starre Lastenverteilung	nicht vorhanden
4.2 Konfigurationsmöglichkeiten	vorhanden – umfangreich	Anzahl Shards/Replicas	Flags per Konsole
Infrastruktur			
5. Dateisystem			
5.1 Import von Daten	möglich – sehr aufwendig	Konsolenbefehl oder Datenbanktreiber	möglich
5.2 Export von Daten	möglich	Konsolenbefehl oder Datenbanktreiber	möglich
5.3 Unterstütze Dateisysteme	unterstützt viele Storage–Backends	Alle vom jeweiligen OS unterstützten	eigenes Dateisystem / SQL-Extension
5.4 Zugriffsmöglichkeiten	HTTP – API, Konsole/Browser	Datenbankqueries	XML -> C# / Queries
5.5 Datenkonsistenz/Verhalten bei Ausfall	Konfigurierbare Datenreplikation, automatische Wiederherstellung, hohe Verfügbarkeit	Manuelle Sicherstellung von Redundanz durch Replica-Sets	Möglichkeit zur Datenreplikation
5.6 Vorteile einer größeren Knotenanzahl	mehr Speicherplatz oder Datensicherheit	Verringerte Last pro Knoten, Datensicherheit	unbekannt
6. Fehlertoleranz			
6.1 Ausfall eines Knotens	Tasks werden neu zugewiesen	Datenverlust	unbekannt
6.2 Konfigurationsmöglichkeiten	nicht vorhanden – automatisch	keine	gering, nur Flags
6.3 Kurzeitiger Verbindungsverlust	Tasks werden neu zugewiesen	keine	unbekannt
7. Monitoring			
7.1 Aktive MapReduce – Jobs	Konsolenausgabe, zukünftig auch über die GUI**	Nicht von anderen Queries zu unterscheiden	Konsolenausgabe
7.2 MapReduce – Jobs in der Queue	nicht vorhanden	Nicht von anderen Queries zu unterscheiden	nicht vorhanden
7.3 Verfügbarkeit von Knoten	wird in der GUI angezeigt	wird angezeigt	Konsolenausgabe
7.4 Arbeitslast von Knoten	wird nicht angezeigt	wird angezeigt	minimal vorhanden
7.5 Aggregierte Sichten	vorhanden	Nur über Drittsoftware	nicht vorhanden
7.6 Logfiles	nocht nicht vorhanden	vorhanden	minimal vorhanden
7.7 Oberflächen/GUI	Riak Control	WebUI für Basisdaten, rest Konsole	nicht vorhanden
7.8 Monitoringmöglichkeiten	vorhanden – überschaubar	Zahlreiche Plugins für Drittsoftware, Eigenes Monitoring für einzelne Knoten	nicht vorhanden
** laut Herstellerangaben			

Erklärung

Hiermit versichern wir, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Stanislaus Biesinger, Michael Pitterle, Mert Canko)