Implementierungstechniken für Hauptspeicherdatenbanksysteme

Michael Schwarz

9. Februar 2018

Zusammenfassung

Mapreduce ist ein von Dean und Ghemawat 2004 vorgestelltes Programmiermodell zur parallelisierten verteilten Berechnung von Operationen auf großen Datenmengen. Das Ziel dieser Arbeit besteht darin eine Adaption von MapReduce für Hauptspeicher-Datenbanksysteme zu entwickeln und dieses in Vergleich zur herkömmlichen Berechnung von Datenbankabfragen zu setzen. Hierzu wurde eine Operation der relationalen Algebra beispielhaft mithilfe von Map-Reduce, sowie auf herkömmlichem Wege implementiert. Eine Herausforderung hierbei stellte die Adaption von MapReduce für den Kontext von Hauptspeicher-Datenbanksystemen dar. Es zeigt sich, dass sich die Verwendung von Map-Reduce für Hauptspeicherdatenbanksysteme nur lohnt, wenn der durch die für MapReduce nötige Partitionierung und Zusammenführung der Daten und Ergebnisse entstehende Overhead durch den Performance-Gewinn durch die Parallelisierung amortisiert wird.

1 Einleitung

1.1 Das Konzept MapReduce

Map-Reduce ist ein Programmier-Modell, eingeführt 2004 von Dean und Ghemawat [1]. Es beinhaltet die zwei Phasen *map* und *reduce*, welche in diesem erstem Kapitel erläutert werden sollen, um so als Grundlage für diese Arbeit zu dienen. Im Grunde wurde Mapreduce entwickelt und eingeführt um die verteilte, parallele Berechnung großer Datenmengen zu ermöglichen, ohne extrem leistungsfähige Rechner zu benötigen. Jeder an diesen Berechnungen beteiligte Computer bekommt eine nicht disjunkte Teilmenge der Daten auf denen die Berechnungen durchgeführt werden sollen.

Zu den großen Vorteilen von MapReduce zählt dass MapReduce ohne tiefgreifende Kenntnisse von verteilter oder paralleler Programmierung verwendet werden kann. Nutzer müssen lediglich zwei Funktionen definieren: Eine map-Funktion und eine reduce-Funktion. Die restliche Verwaltung des Datentransfers, der Ausfallsicherung, der Parallelisierung wird von MapReduce übernommen.

Zwischen den beiden nutzerspezifizierten Funktionen steht noch eine dritte Phase, die als Schnittstelle zwischen diesen beiden Funktionen begriffen werden kann: Die Shuffle-Funktion. Diese wird jedoch von MapReduce bereitgestellt und muss nicht vom Nutzer spezifiziert werden. Jede dieser drei Funktionen stellt eine Transformation der ursprünglichen Daten dar. Diese Transformationen können wie folgt formal spezifiziert werden:

$$\{(k, v)\}^* \to \{(l, u)\}^* \to \{(l, u^*)\}^* \to \{(l, u)\}^*$$

Dabei wird jede der Transformationen durch eine der drei für MapReduce cha-

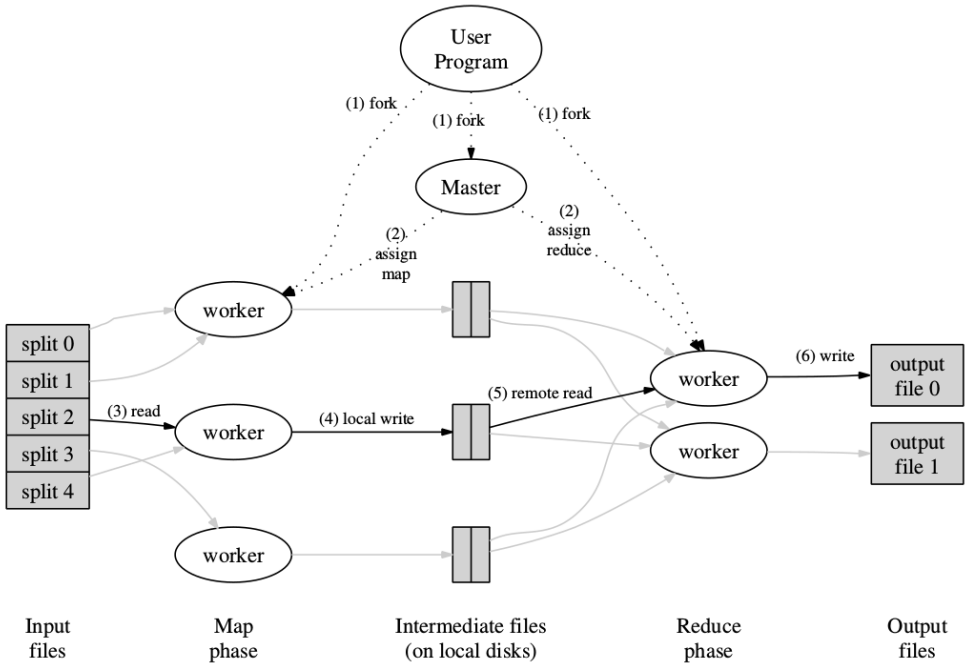

Abbildung 1: Ablauf von MapReduce, Quelle: [1]

rakteristischen Funktionen durchgeführt. Diese werden im Folgenden als Phasen bezeichnet.

1.1.1 Die Map-Phase

In der Map-Phase führt jeder Computer die gewünschten Berechnungen auf der ihm zugeteilten Teilmenge der Daten aus. Diese liegen zu Beginn der Map-Phase auf dem Festplattenspeicher des jeweiligen Computers. Die Map-Funktion bekommt bei jedem Aufruf ein Schlüssel-Wert-Paar $\{(k, v)\}$ und generiert daraus wiederum ein Schlüssel-Wert-Paar $\{(l, u)\}$. Dieses Ergebnis wird auf der Festplatte zwischengespeichert. Formal kann die Map-Phase also definiert werden als:

$$\{(k, v)\}^* \rightarrow \{(l, u)\}^*$$

.

1.1.2 Die Shuffle-Phase

In der Shuffle-Phase werden alle Werte, die zu dem gleichen Schlüssel gehören gesammelt und diesem Schlüssel zugeordnet:

$$\{(l, u)\}^* \rightarrow \{(l, u^*)\}^*$$

1.1.3 Die Reduce-Phase

Jedes der Paare Schlüssel-Werteliste, die Resultat der Shuffle-Phase sind wird in der Reduce-Phase auf ein Paar Schlüssel-Wert reduziert:

$$\{(l, u^*)\}^* \rightarrow \{(l, u)\}^*$$

1.2 Verwandte Arbeiten

Abouzeid et al. stellten 2009 Hadoop vor, eine Hybride Lösung aus DBMS und MapReduce [2]. Hadoop stellt dabei insofern einen Hybriden aus DBMS und MapReduce dar, als dass sich Hadoop MapReduce zu nutze macht um auf einem parallelen verteilten Datenbanksystem parallel Abfragen durchzuführen.

Gruska und Martin [5] verglichen verschiedene Ansätze MapReduce und Datenbanksysteme zu vereinen und untersuchten Hadoop's DBInputFormat, welches für Hadoop benötigt wird.

Lee et al haben mit YSmart einen SQL-zu-MapReduce-Übersetzer entwickelt, der in der Lage ist Verwandtschaften zwischen Anfragen zu berücksichtigen und so die Performanz zu erhöhen [4].

Shankar und Dijcks hingegen haben gezeigt, wie man mit *Oracle Table Functions* MapReduce in einer Oracle Datenbank hinzufügen kann [3].

2 Operatoren der relationalen Algebra mit MapReduce

Zur Untersuchung des Nutzens der Implementierung von Operationen der relationalen Algebra werden in diesem Kapitel grundsätzliche Abbildungen verschiedener dieser Operationen auf MapReduce vorgestellt. Der Schwerpunkt liegt hierbei bei den Aggregatsfunktionen, da diese eine zweistufige Abarbeitung der Anfrage forcieren.

2.1 Selektion

Für die Selektion ist grundsätzlich gesehen nur eine Map-Phase nötig. Hierbei werden durch die Map-Funktion nur diejenigen Einträge emittiert, die das Selektions-Kriterium erfüllen. Eine Reduce-Phase ist hier dann nicht mehr notwendig, da die Selektion nach der Map-Phase abgeschlossen ist.

2.2 Projektion

Die Projektion erfordert wie die Selektion keine Reduce-Phase. Während der Map-Phase wird eine Kopie jedes Eintrages der für die Projektion verwendeten Relation erstellt, wobei die Kopie nur die anzuzeigenden Spalten beinhaltet.

2.3 Aggregatsfunktionen

Aggregatsfunktionen nehmen im Vergleich zu Selektion und Projektion eine besondere Stellung ein, da sie eine zweistufige Bearbeitung der Anfrage gut ermöglichen. In der ersten Stufe wird hierbei eine Gruppierung im Sinne des *group-by*-Operators durchgeführt, in der zweiten Stufe auf den erhaltenen Gruppierungen die jeweilige Aggregatsfunktion.

2.3.1 Anzahl

Die Berechnung der Anzahl an Vorkommen verschiedener Werte ist ein gängiges Beispiel zur Verwendung von Map-Reduce [1]. In der Map-Phase wird jeweils der Wert des Attributs dessen Vorkommen gezählt werden soll zum Schlüssel, der zugehörige Wert ist 1.

In der Reduce-Phase wird dann die Summe aller Vorkommen gezählt. Das Resultat ist dann eine Zuordnung Wert-Anzahl.

2.3.2 Summe

Zur Berechnung der Summe der Werte eines Attributes wird zunächst in der Map-Phase die Gruppierung durchgeführt.

In der Reduce-Phase wird für jede der Gruppen die Summe des zu summierenden Attributes ermittelt.

2.3.3 Durchschnitt

Der Durchschnitt kann durch die Bildung der Summe, sowie der Anzahl über der gleichen Relation berechnet werden. Hierzu wird die Summe durch die Anzahl geteilt. Für dieses Anwendungsszenario ist die Verwendung einer dedizierten Map-Reduce-Implementierung somit nicht nötig.

2.3.4 Maximum, Minimum

Das Auffinden des Minimums erfolgt analog zum Maximum. Der Wert findet sich jedoch am Anfang der Map.

3 Adaption von MapReduce für Hauptspeicherdatenbanksysteme

Während MapReduce ursprünglich entworfen wurde um große Datenmengen, parallel und verteilt zu verarbeiten, wobei diese Daten auf den Festplatten der Rechenmaschinen zwischengespeichert werden, arbeiten Hauptspeicher-Datenbanken auf dem Hauptspeicher eines Rechners.

Um MapReduce für eine Hauptspeicherdatenbank zu adaptieren, sind einige Anpassungen des Konzeptes notwendig: Anstatt die Rechenlast auf verschiedene Computer zu verteilen wird die Parallelisierung durch mehrere Kerne erreicht werden. Die Daten werden nicht verteilt gehalten, sondern verbleiben im Hauptspeicher. Somit müssen Daten nicht auf Festplatten geschrieben oder von ihnen gelesen werden.

Die Implementierung der Adaption von MapReduce für Hauptspeicherdatenbanksysteme sieht drei Basisklassen vor:

- MapReduce
- Mapper
- Reducer

Die Klasse MapReduce übernimmt alle Verwaltungsaufgaben des MapReduce-Prozesses. Dies beinhaltet die Partitionierung der Daten für die jeweiligen Mapper, das Zusammenführen der Ergebnisse der Mapper (*shuffle*), das Sammeln der Ergebnisse der Reducer, sowie die Verwaltung der jeweiligen Threads.

Die Map-Funktion verarbeitet bei jeder Ausführung einen Datensatz. Sie bekommt als Key den Index des Argumentes nach dem Gruppiert werden soll und als Wert einen Pointer auf den zu bearbeitenden Datensatz. Der entsprechende Pseudocode ist in Abbildung 2 zu sehen.

```
void map(int key, Record* value) {
   emit(value[key],value);
}
```

Abbildung 2: Map-Funktion für GroupBy in Pseudocode

Die Reduce-Funktion ermittelt für jede der ermittelten Gruppierungen die Summe der Werte des zu summierenden Attributs. Dieser Vorgang ist in Abbildung 3 zu sehen.

```
void reduce(int key, std::vector<Record*> r) {
int sum = 0;
for(RefRelation::iterator it = r.begin(); it < r.end(); ++it) {
sum += (**it)[col];
}
collect(key,sum);
}
```

Abbildung 3: Reduce-Funktion für GroupBy in Pseudocode

4 Evaluation und Ergebnis

Im Zuge der Evaluation wurden Map-Reduce für die Aggregatsfunktion Summe mit C++ implementiert. Zur Vereinfachung wurde hierbei eine Relation die nur aus Attributen mit ganzzahligem Datentyp besteht angenommen.

Vergleichsimplementierung

Als Vergleich diente eine nicht parallelisierte Vergleichsimplementierung, die sich sequentiell durch die Daten arbeitet. Für jeden Eintrag wird dabei der Wert des zur Gruppierung dienenden Attributs als Schlüssel für eine Hashmap benutzt. Der zugehörige Wert wird jeweils um den Wert des zu summierenden Attributs erhöht.

Testumgebung

Die Performance-Tests wurden auf einem System mit *Intel® HM86 Express Chipset* ausgestattetem System mit Intel® Core™ i7 4700HQ Prozessor mit 4 physischen Kernen und mit hyperthreading 8 logischen, sowie 16gb DDR3-SDRAM mit einer Taktung von 1600MHz durchgeführt.

Ergebnis

Wie in Abbildung 4 zu sehen ist, kann eine Steigerung der Anzahl an Threads zu einer Steigerung der Leistung führen, solange die Anzahl an Threads nicht die Anzahl physisch verfügbarer Kerne übersteigt. Nicht nur steigt dann die Laufzeit wieder an, sondern erhöht sich auch die Standardabweichung enorm.

Die Tests zeigten in diesem Anwendungsfall keine Performance-Steigerung durch MapReduce. Viel mehr benötigte die Implementierung mittels MapReduce wesentlich mehr Laufzeit.

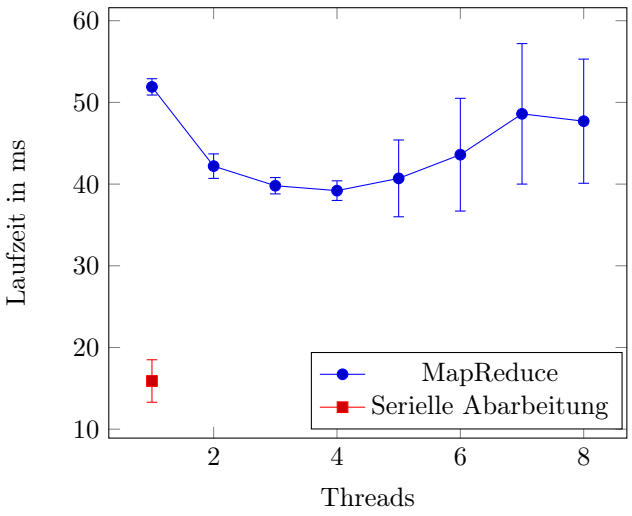

Abbildung 4: Laufzeiten von MapReduce für verschiedene Anzahlen Threads

5 Ausblick

Es wurde gezeigt, dass es möglich ist, MapReduce auch für ein Hauptspeicherdatenbanksystem zu implementieren. Auf diese Art und Weise könnte es einerseits Nutzern einer Hauptspeicherdatenbank ermöglicht werden ihre Abfragen mittels einer Map-Funktion und einer Reduce-Funktion zu spezifizieren, ohne auf SQL zurückgreifen zu müssen, während es andererseits nicht ausgeschlossen werden kann, dass es Anwendungsfälle gibt, in denen die Parallelisierung mit MapReduce einen Performance-Gewinn bringt. Für einen solchen Anwendungsfall muss der Performance-Gewinn durch die Parallelisierung den Aufwand für die Partitionierung der Daten sowie die Reorganisation der Daten übersteigen.

Aus der entwickelten Implementierung von GroupBy mit MapReduce kann eine generelle Implementierung von MapReduce für Hauptspeicherdatenbanksysteme abstrahiert werden.

Literatur

[1] Jeffrey Dean, Sanji Ghemawat. MapReduce: simplified data processing on large clusters. In OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, pages 137–150, Berkeley, CA, USA, 2004. USENIX Association.

[2] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, Alexander Rasin. In Proceedings of the Conference on Very Large Databases (Lyon, France, 2009); http://db.cs.yale.edu/hadoopdb/: HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads.

[3] Shrikanth Shankar, Jean-Pierre Dijcks. In-Database Map-Reduce

[4] Rubao Lee, Tian Luo, Yin Huai, Fusheng Wang, Yongqiang He, Xiaodong Zhang. YSmart: Yet Another SQL-to-MapReduce Translator

[5] Natalie Gruska, Patrick Martin. Integrating MapReduce and RDBMS