UNIVERSITÄT DÜSSELDORF
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Prof. Dr. Michael Schöttner
Florian Klein

Abschlussprojekt
„Verteilte und Parallele Systeme“

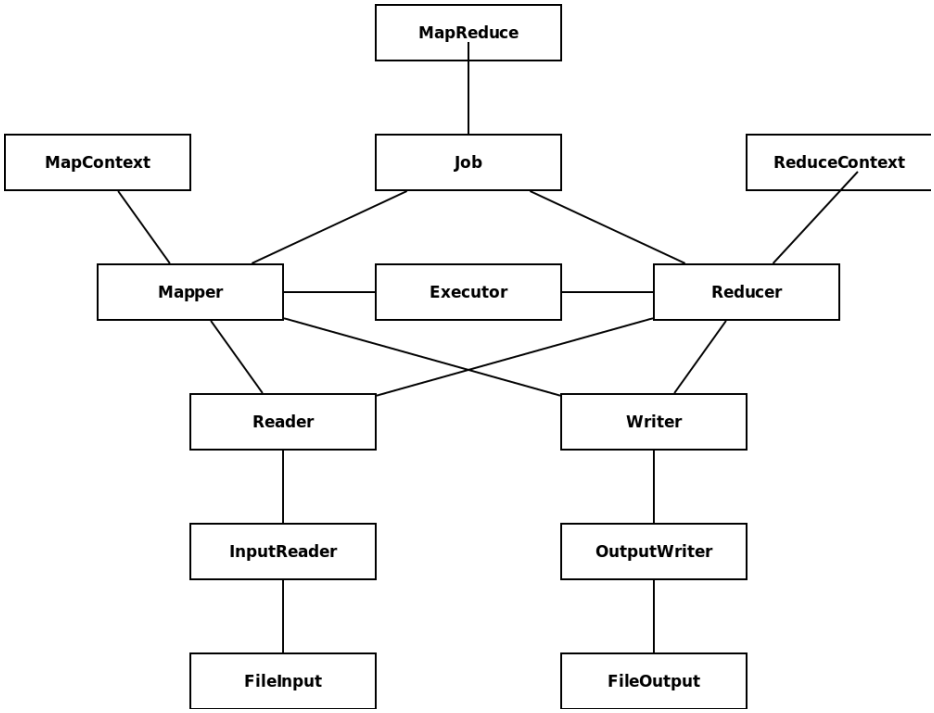
Abgabe Aufgabe 1-4 bis Sonntag, den 29.06.2013 um 23:59 Uhr
Abgabe Aufgabe 5-6 bis Sonntag, den 06.07.2013 um 23:59 Uhr

MapReduce-Framework

Ihre Aufgabe ist es ein MapReduce-Framework zu implementieren, dass es erlaubt beliebige MapReduce-Jobs mehrfädig auf dem lokalen Rechner auszuführen.
Zu diesem Zweck wird eine grundlegende Implementierung zur Verfügung gestellt, die in mehreren Schritten erweitert werden muss. Abschließend müssen mehrere einfache Anwendungen realisiert werden.

Die Grundidee von MapReduce besteht darin, parallele Anwendungen durch Implementierung der beiden Methoden *map()* und *reduce()* umsetzen zu können. Alle anderen Aufgaben, wie zum Beispiel das Ausführen von Worker-Threads oder die Verteilung der Daten, übernimmt das Framework.
Die Bearbeitung eines MapReduce-Jobs erfolgt verteilt auf mehreren Map- und Reduce-Tasks, welche mehrere Phasen durchlaufen. Zunächst werden die Eingabedaten gleichmäßig auf die Map-Tasks verteilt und die Map-Phase beginnt. Dort werden die zugeteilten Datenbereiche eingelesen und durch die Map-Tasks verarbeitet. Die entstandenen Zwischenergebnisse werden von jedem Mapper zunächst einem bestimmten Reduce-Task zugeordnet (Partitionierung) und die einzelnen Teile anschließend sortiert. Diese sortierten Teile werden dann jeweils in einer eigenen Datei gespeichert, so dass am Ende der Map-Phase für jede Kombination aus Map-Task und Reduce-Task ein Zwischenergebnis vorliegt. In der darauffolgenden Reduce-Phase lesen die Reduce-Tasks Ausgaben ihrer zugehörigen Partitionen aus allen Mapper-Ausgaben ein. Die Vorsortierung der Map-Tasks wird dabei genutzt, um alle zu einem Schlüssel zugehörigen Werte der Reduce-Funktion zuzuführen. Endergebnisse eines Reduce-Tasks werden in jeweils einer eigenen Datei gespeichert.

Das folgende Schaubild zeigt einen Überblick über das MapReduce-Framework und seine Komponenten. Die wichtigsten Komponenten werden im Rahmen der Teilaufgaben im Detail erläutert.



Aufgabe 1: Einlesen von Daten 5 Punkte

Beim MapReduce-Programmiermodel werden an verschiedenen Stellen Daten eingelesen und verarbeitet. Das Einlesen von einzelnen Textzeilen aus einer Datei wird durch die Klassen *FileInput* und *InputReader* ermöglicht.
Alle Klassen dieser Aufgabe sind im Package *vps.mapreduce.reader* zu finden.

Aufgabe 1.1: KeyValueReader 1 Punkt

In der Regel werden aber nicht einzelne Textzeilen benötigt, sondern Schlüssel-Wert Paare. Eine Textzeile entspricht in diesem Falle einem solchen Schlüssel-Wert Paar. Schlüssel und Wert sind dabei durch ein definiertes Zeichen (*Configuration.KEY_VALUE_SEPERATOR*) voneinander getrennt.
Ihre Aufgabe ist es die Klasse *KeyValueReader* zu implementieren, die eine Textzeile von einem untergeordnetem *Reader* entgegen nimmt und in ein Schlüssel-Wert Paar aufspaltet. Falls in einer Zeile nur ein Wert steht, soll die Zeilennummer als Schlüssel verwendet werden. Die Klasse existiert bereits und muss nur noch von Ihnen gefüllt werden.

- **Implementieren Sie die Klasse *KeyValueReader***

Aufgabe 1.2: IterableValueReader 2 Punkte

Für die Ausführung des Reduce-Tasks ist es außerdem notwendig zu einem Schlüssel alle möglichen Werte in einer iterierbaren Datenstruktur zu erhalten. Zu diesem Zweck dient die Klasse *IterableValueReader*. Sie liest hintereinander folgende Schlüssel-Wert Paare aus und gibt den Schlüssel zusammen mit einer Referenz auf eine iterierbare Datenstruktur mit den Werten weiter. Dazu implementiert sie die Schnittstelle *Iterable* und *Iterator* (Package java.util) für die zugehörigen Werte.
Ihre Aufgabe ist es die leeren Methodenrümpfe in der Klasse *IterableValueReader* zu füllen.

- **Implementieren Sie die Klasse *IterableValueReader***

Aufgabe 1.3: MergingReader 2 Punkte

Beim Einlesen der Daten für die Reducer muss in der Regel der Inhalt mehrerer Dateien berücksichtigt und zusammengeführt werden. Der Inhalt der Dateien ist nach Schlüsseln sortiert und die Sortierung muss auch beim Zusammenführen aller Dateien weiterhin erhalten bleiben. Dies garantiert unter anderem, dass Zeilen mit dem gleichen Schlüssel hintereinander eingelesen werden. Die Klasse *MergingReader* verwaltet ein Array von *Readern* und führt ihre Daten zusammen.
Ihre Aufgabe ist es die leeren Methodenrümpfe in der Klasse *MergingReader* zu füllen.

- **Implementieren Sie die Klasse *MergingReader***

Hinweis: Sie können die interne Klasse BufferedReader verwenden, um die Eingabe der verschiedenen Reader mit Hilfe der compareTo()-Methode zu vergleichen.

Aufgabe 2: Schreiben von Daten 3 Punkte

Nach der Verarbeitung durch den Map- bzw. Reduce-Task werden die Daten weg geschreiben. Das Schreiben von einzelnen Textzeilen in eine Datei wird durch die Klassen *FileOutput* und *OutputWriter* ermöglicht.
Alle Klassen dieser Aufgabe sind im Package *vps.mapreduce.writer* zu finden.

Aufgabe 2.1: KeyValueWriter 1 Punkt

Die Klasse *KeyValueWriter* wird benutzt um ein Schlüssel-Wert Paar (beides Strings) in eine Textzeile umzuwandeln und diese an einen darunter liegenden Writer zu übergeben. Als Trennzeichen zwischen Schlüssel und Wert soll das Zeichen *Configuration.KEY_VALUE_SEPERATOR* verwendet werden.

- **Implementieren Sie die Klasse *KeyValueWriter***

Aufgabe 2.2: SplittingWriter 2 Punkte

Die Ausgabe eines Mappers erfolgt in der Regel in mehrere Dateien (eine für jeden Reducer). Die Klasse *SplittingWriter* teilt dabei die Ausgabe eines Mapper auf mehrere *Writer* auf. Die Zuteilung des *Writers* soll auf Basis des übergebenden Wertes ausgeführt werden.

- **Implementieren Sie die Klasse *SplittingWriter***

Aufgabe 3: Mapper und Reducer 2 Punkte

Die Klassen Mapper und Reducer im Package vps.mapreduce.core repräsentieren einen Map- bzw. Reduce-Task und stellen die map()- bzw. reduce()-Methode zur Verfügung.

Aufgabe 3.1: Mapper 1 Punkt

Der Map-Task verfügt über einen *Reader* zum Einlesen von Daten, einen *MapContext* zum Ablegen von Schlüssel-Wert Paaren und einen *Writer* zum Schreiben von Daten.
Darüber hinaus existiert die Methode *execute()*, die den Map-Task ausführt. Die Methode liest ein Schlüssel-Wert Paar mit Hilfe des *Readers* ein und verarbeitet dieses mit der *map()*-Methode. Dieser Vorgang wird so lange wiederholt, bis alle Daten verarbeitet wurde. Anschließend werden die Daten mit der *flush()*-Methode der Klasse *MapContext* und dem *Writer* geschrieben. Zu aller Letzt müssen noch der *Reader* und *Writer* geschlossen werden.
Ihre Aufgabe ist es die *execute()*-Methode zu implementieren.

- **Implementieren Sie die *execute()*-Methode der Klasse *Mapper***

Aufgabe 3.2: Reducer 1 Punkt

Ähnlich wie der Map-Task verfügt der Reduce-Task über einen *Reader* zum Einlesen von Daten, einen *ReduceContext* zum Ablegen von Schlüssel-Wert Paaren und einen *Writer* zum Schreiben von Daten.
Darüber hinaus existiert die Methode *execute()*, die den Reduce-Task ausführt. Die Methode liest ein Schlüssel-Wert Paar mit Hilfe des *Readers* ein und verarbeitet dieses mit der *reduce()*-Methode. Dieser Vorgang wird so lange wiederholt, bis alle Daten verarbeitet wurde. Anschließend werden die Daten mit der *flush()*-Methode der Klasse *ReduceContext* und dem *Writer* geschrieben. Zu aller Letzt müssen noch der *Reader* und *Writer* geschlossen werden.
Ihre Aufgabe ist es die *execute()*-Methode zu implementieren.

- **Implementieren Sie die *execute()*-Methode der Klasse *Reducer***

Aufgabe 4: Ausführung des MapReduce-Frameworks 5 Punkte

Das Framework soll von der Konsole mit dem folgenden Befehl aufzurufen sein:

java vps.mapreduce.MapReduce <job> <infile> <temp_prefix> <out_prefix>

- Der Parameter <*job*> bestimmt den MapReduce-Job der ausgeführt werden soll. Alle MapReduce-Jobs müssen das Interface *Job* (Package *vps.mapreduce.core*) implementieren und im Package *vps.mapreduce.jobs* liegen. Der Parameter muss dabei genau dem Klassennamen entsprechen (Groß- und Kleinschreibung beachten). Soll der MapReduce-Job *vps.mapreduce.jobs.TestJob* ausgeführt werden, muss für <*job*> die Zeichenkette „TestJob“ angegeben werden.
- Der Parameter <*infile*> enthält den Dateinamen der initialen Eingabedatei.
- Der Parameter <*temp_prefix*> enthält das Dateinamen-Prefix für die temporären Dateien (zwischen Map- und Reduce-Task).

- Der Parameter <*out_prefix*> enthält das Dateinamen-Prefix der Ausgabedateien.

Ihre Aufgabe ist es die *main()*-Methode in der Klasse *MapReduce* (Package *vps.mapreduce*) zum Starten des MapReduce-Frameworks zu implementieren. Dafür müssen zunächst die übergebenen Parameter ausgewertet werden und eine Instanz des auszuführenden Jobs erzeugt werden. Anschließend müssen jeweils eine definierte Anzahl an Mappern und Reducern erzeugt und ausgeführt werden. Ein Mapper bzw. Reducer wird über die Methode *createMapper()* bzw. *createReducer()* der Klasse Job erzeugt. Die benötigten *Reader* und *Writer* werden über die statischen Methoden *Configuration.createMapReader()*, *Configuration.createMapWriter()*, *Configuration.createReduceReader()*, *Configuration.createReduceWriter()* erstellt. Die Ausführung der Mapper bzw. Reducer wird mit der statischen Methode *Executor.execute()* ermöglicht. Die Anzahl der Mapper, Reducer und die Thread Anzahl für den *Executor* können der Klasse *Configuration* entnommen werden.

- **Implementieren Sie die *main()*-Methode der Klasse *MapReduce***

Hinweis: Im Package vps.mapreduce.jobs existiert schon die Klasse WordCount. Diese kann genutzt werden, um zu Überprüfen, ob das MapReduce-Framework korrekt arbeitet. Dazu finden Sie auch eine Datei mit einem WordCount-Beispiel „example_wc.txt“ und eine Datei mit der korrekten Ausgabe „out_wc.txt“.

Aufgabe 5: MapReduce-Job CommonFriends — 4 Punkte

Eine typische Funktion bei sozialen Netzwerken ist die Anzeige der gemeinsamen Freunde. Dabei ist eine Freundschaft eine bidirektionale Beziehung. Die Berechnung der gemeinsamen Freunde in großen Netzwerken ist relativ aufwändig, weswegen sie nicht bei jeder Anfrage, sondern nur in größeren Abständen durchgeführt wird (beispielsweise einmal am Tag).
In dieser Aufgabe sollen Sie die Berechnung der gemeinsamen Freunde mit Hilfe des MapReduce-Frameworks durchführen. Im Package *vps.mapreduce.jobs* existiert bereits eine Klasse *CommonFriends*, mit einer inneren Klasse *CFSet* zur Verwaltung von Mengen. Sowohl Schlüssel als auch Wert der verwendeten Schlüssel-Wert Paare sollen vom Typ *CFSet* sein.

Hinweis: Auf der Seite http://www.stevekrenzel.com/finding-friends-with-mapreduce wird ein möglicher Algorithmus zur Berechnung sehr gut an einem Beispiel erklärt.

Aufgabe 5.1: CFSet Vergleich — 1 Punkt

Für die Verwendung von *CFSet* als Schlüssel, muss die Klasse das Interface *java.lang.Compareable* implementieren, um zwei Objekte vom Typ *CFSet* miteinander vergleichen zu können.

- **Implementieren Sie die *compareTo()*-Methode der Klasse *CFSet***

Aufgabe 5.2: Reader und Writer — 1 Punkt

Da Schlüssel und Wert keine *Strings* sind, sondern vom Typ *CFSet*, müssen ein eigener *Reader* (*CFSetReader*) und *Writer* (*CFSetWriter*) implementiert werden.

- **Implementieren Sie die Klassen *CFSetReader* und *CFSetWriter***

Aufgabe 5.3: Mapper und Reducer — 2 Punkte

Zuletzt müssen noch eine Mapper (*CFMapper*) bzw. Reducer (*CFReducer*) Klasse implementiert und die Implementierung der Klasse *CommonFriends* vervollständigt werden.

- **Implementieren Sie die Klassen *CFMapper, CFReducer* und *CommonFriends***

Aufgabe 6: Jar-Archiv — 1 Punkt

Erstellen Sie ein ausführbares Jar-Archiv mit Binär- und Quellcode.

Hinweis: Die Aufgaben 1-4 müssen bis Sonntag, den 29.06.2014 um 23.59 Uhr bei Herrn Florian Klein per Mail abgegeben werden (Florian.Klein@uni-duesseldorf.de)
Hinweis: Die Aufgaben 5-6 müssen bis Sonntag, den 06.07.2014 um 23.59 Uhr bei Herrn Florian Klein per Mail abgegeben werden (Florian.Klein@uni-duesseldorf.de)
Hinweis: Sollten Sie die Aufgaben 1-4 nicht vollständig gelöst bekommen, stellen wir ab dem 30.06.2014 eine Musterlösung zur Verfügung, mit der Sie die letzten beiden Aufgabenteile erledigen können.
Hinweis: In den Übungen haben Sie die Möglichkeit Fragen zum Abschlussprojekt zu stellen.
Hinweis: Das Ergebnis des Abschlussprojektes fließt in die Endnote des Moduls ein.