Aufgabenblatt 7
Web-Scale Data Management

- Abgabetermin: **-** (dieses Blatt wird weder abgegeben noch bewertet)
- Dieses Aufgabenblatt enthält einige Übungen zum Thema Web Scale Data Management.

Aufgabe 1: MapReduce

Ein optisches Messverfahren erfasst die Oberfläche von Objekten als Menge von 3D-Ortsvektoren $(x, y, z)^T$ und speichert diese in der Datenbanktabelle $S(\underline{vectorId}, \underline{dimension}, value)$ ab. Durch Störeinflüsse werden beim Messen nicht alle Ortsvektoren vollständig erfasst.

Für eine grafische Simulation werden nun die Vektorlängen *length* aller **vollständig erfasster** Ortsvektoren benötigt, $length = \sqrt{x^2 + y^2 + z^2}$. Aufgrund der großen Datenmenge soll die Berechnung auf einem MapReduce-Cluster erfolgen.

a) Als Ausgabe soll eine Tabelle $T(\underline{vectorId}, length)$ erzeugt werden, in der für jeden vollständig erfassten Ortsvektor die Vektorlänge steht. Die Ausgabe wird partitioniert auf den verschiedenen Nodes ins verteilte Dateisystem geschrieben. Lösen Sie die Aufgabe mit nur einem MapReduce-Job. Verwenden Sie auch die in der Übung vorgestellte Funktion `combine`, um Netzwerklast zu reduzieren. Beschreiben Sie Ihr Vorgehen kurz in wenigen Sätzen. Erstellen Sie den Pseudocode für `map`, `combine` und `reduce`. **8 P**

b) Die Abbildung zeigt beispielhaft eine Verteilung von S auf drei MapReduce-Nodes. Zeigen Sie für dieses Beispiel die Ausgabe der einzelnen Phasen (`map`, `combine` und `reduce`) Ihres MapReduce-Jobs auf jeder der drei Nodes. **3 P**

id	*dim*	*val*
1	x	2
2	z	4
1	y	3
3	y	2
4	y	4

id	*dim*	*val*
3	z	4
4	z	2
2	x	2
2	y	4

id	*dim*	*val*
5	y	3
3	x	4
4	x	4

Beginn Musterlösung

a) Die Map-Funktion emittiert als Schlüssel die ID des Vektors und als Wert jeweils den quadrierten Dimensionswert. **2 P**

```
map (key, tuple) {
  emit (tuple.vectorid, tuple.value * tuple.value);
}
```

Die Combine-Funktion erhält als Schlüssel die ID eines Vektors und als Wert die Liste der zugehörigen quadrierten Dimensionswerte, die dann aufsummiert werden. Für vollständige Vektoren wird die Vektorlänge ins verteilte Dateisystem geschrieben. Die Kennzahlen der unvollständigen Vektoren werden über das Netzwerk zur Reduce-Funktion geschickt. **3 P**

```
combine (key, values) {
  s = 0;
```

```
  foreach (values as v)
    s += v;
  if (values.size() == 3)
    write (key, sqrt(s));
  else
    emit (key, {'count' => values.size(), 'sum' => s});
}
```

Die Reduce-Funktion schreibt für vollständige Vektoren die Vektorlänge ins verteilte Dateisystem. **3 P**

```
reduce (key, tuples) {
  c = 0;
  s = 0;
  foreach (tuples as t) {
    c += t.count;
    s += t.sum;
  }
  if (c == 3)
    write (key, sqrt(s));
}
```

Hinweis: Wir benutzen hier eine Funktion `write`, um das Schreiben ins verteilte Dateisystem darzustellen.

b) **Phase 1 - Ausgabe (map)**

key	*value*
1	4
2	16
1	9
3	4
4	16

key	*value*
3	16
4	4
2	4
2	16

key	*value*
5	9
3	16
4	16

Phase 2 - Ausgabe (combine)

key	*value*
1	{2,13}
2	{1,16}
3	{1, 4}
4	{1,16}

key	*value*
3	{1,16}
4	{1, 4}
2	{2,20}

key	*value*
5	{1, 9}
3	{1,16}
4	{1,16}

Phase 3 - Ausgabe (reduce)

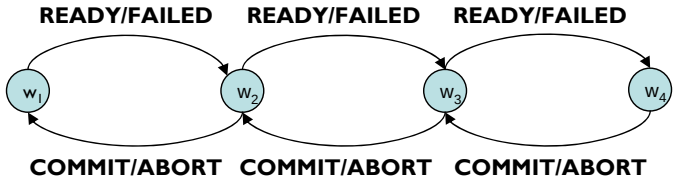
key	*value*
2	6
3	6
4	6

Ende Musterlösung

Aufgabe 2: Two Phase Commit (Bonusaufgabe)

In der Vorlesung wurde eine hierarchische Organisationsstruktur (ein Koordinator und mehrere untergeordnete Worker) beim 2PC-Prototoll beschrieben. Es ist auch möglich, die in der Abbildung gezeigte lineare Organisationsstruktur vorzunehmen.

Hierbei ist kein ausgezeichneter Koordinator erforderlich. In der ersten Phase reichen die Worker ihren eigenen Status und den der linken Nachbarn von *links nach rechts* weiter, nachdem sie einen entsprechenden Statusbericht von links bekommen haben. Der letzte Worker in der Reihe – hier Worker w_4 – trifft die Entscheidung und reicht sie nach links weiter.



Entwickeln Sie das 2PC-Protokoll – analog zur Vorlesung – für diese lineare Anordnung der Worker als Pseudocode und beschreiben Sie kurz das Vorgehen. **4**
Bonuspunkte

Beginn Musterlösung

Commit Request Phase	**INPUT**
	msg=READY or msg = EndTransaction;
	ACTION
	LocalState := CheckCommit();
	Log(LocalState); // READY / FAILED
	if hasNextWorker()
	 send LocalState to nextWorker;
	else
	 if LocalState = READY
	 send COMMIT to self;
	 elseif LocalState = FAILED
	 send ABORT to self;
	 fi;
	fi;
		INPUT
		msg = FAILED;
		ACTION
		Log(FAILED);
		if hasNextWorker()
		 send FAILED to nextWorker;
		else
		 send ABORT to self;
		fi;

Commit Phase	**INPUT**
	msg = COMMIT;
	ACTION
	Log(COMMIT);
	CommitLocalTransaction();
	FreeLocalResources();
	if(hasPreviousWorker())
	 send COMMIT to previousWorker;
	fi;
		INPUT
		msg = ABORT;
		ACTION
		Log(ABORT);
		AbortLocalTransaction();
		FreeLocalResources();
		if(hasPreviousWorker())
		 send ABORT to previousWorker;
		fi;

Ende Musterlösung