

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 $S(\text{vectorId}, \text{dimension}, \text{value})$ 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) Erzeuge als Ausgabe eine Tabelle $T(\text{vectorId}, \text{length})$, in der für jeden vollständig erfassten Ortsvektor die Vektorlänge steht. Die Ausgabe wird partitioniert auf den verschiedenen Nodes ins verteilte Dateisystem geschrieben. Löse die Aufgabe mit nur einem MapReduce-Job. Verwende auch die in der Vorlesung vorgestellte Funktion *combine*, um Netzwerklast zu reduzieren. Beschreibe dein Vorgehen kurz in wenigen Sätzen. Erstelle den Pseudocode für *map*, *combine* und *reduce*.

Lösung

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) Die Abbildung zeigt beispielhaft eine Verteilung von S auf drei MapReduce-Nodes. Zeige für dieses Beispiel die Ausgabe der einzelnen Phasen deines MapReduce-Jobs auf jeder der drei Nodes. 3 P

Eingabe (3 Datanodes)

<i>id</i>	<i>dim</i>	<i>val</i>
1	x	2
2	z	4
1	y	3
3	y	2
4	y	4

<i>id</i>	<i>dim</i>	<i>val</i>
3	z	4
4	z	2
2	x	2
2	y	4

<i>id</i>	<i>dim</i>	<i>val</i>
5	y	3
3	x	4
4	x	4

Phase 1 - Ausgabe (map)

<i>key</i>	<i>value</i>
1	4
2	16
1	9
3	4
4	16

<i>key</i>	<i>value</i>
3	16
4	4
2	4
2	16

<i>key</i>	<i>value</i>
5	9
3	16
4	16

Phase 2 - Ausgabe (combine)

<i>key</i>	<i>value</i>
1	{2,13}
2	{1,16}
3	{1, 4}
4	{1,16}

<i>key</i>	<i>value</i>
3	{1,16}
4	{1, 4}
2	{2,20}

<i>key</i>	<i>value</i>
5	{1, 9}
3	{1,16}
4	{1,16}

Phase 3 - Ausgabe (reduce)

<i>key</i>	<i>value</i>
2	6
3	6
4	6

Aufgabe 3: Two Phase Commit (2PC)

Commit Request Phase	<p>INPUT msg=READY or msg = EndTransaction;</p> <p>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;</p>	<p>INPUT msg = FAILED;</p> <p>ACTION Log(FAILED); if hasNextWorker() send FAILED to nextWorker; else send ABORT to self; fi;</p>
Commit Phase	<p>INPUT msg = COMMIT;</p> <p>ACTION Log(COMMIT); CommitLocalTransaction(); FreeLocalResources(); if(hasPreviousWorker()) send COMMIT to previousWorker; fi;</p>	<p>INPUT msg = ABORT;</p> <p>ACTION Log(ABORT); AbortLocalTransaction(); FreeLocalResources(); if(hasPreviousWorker()) send ABORT to previousWorker; fi;</p>