Verteilte Berechnungsplattformen in der Praxis

Lukas Pietzschmann
Universität Ulm - Institut für Verteilte Systeme

universität ulm

ÜBUNGSBLATT 1: In-Memory Processing

Fällig am 22.01.2023

Hinweise

1. Der Hadoop-Stack stellt einige Web-UIs zur Verfügung:

   9870 HDFS-Übersicht

   8088 YARN-Übersicht

   4040 Spark-Übersicht

   Wollt ihr diese Webseiten nutzen, müsst ihr zuerst die jeweiligen Ports des Master-Knotens in der BW-Cloud freigeben (Network → Security Groups → Manage Rules). Die Oberflächen sind dann über `http://<master-ip>:<port>` aufrufbar.

2. Eine ausführliche Dokumentation Sparks python-API findet ihr unter `https://spark.apache.org/docs/latest/api/python/reference/pyspark.html`

Aufgabe 0 Setup

Wie schon in der ersten Übung zu Batch Processing beinhaltet das Archiv eine `inventory.ini` und `ansible-playbook.yml` Datei. In ersterer müsst ihr wieder die IPs eurer Knoten eintragen. Zur einfacheren Verwendung des Playbooks habe ich ein Makefile geschrieben. Folgende Targets könnt ihr nutzen:

all Damit wird das komplette Playbook ausgeführt. `all` sollte beim ersten Ausführen verwendet werden um alle Knoten korrekt zu initialisieren.

deploy Dieses Target überspringt alle initialisierungs-Schritte und führt direkt den aktuellen Code aus. Nach dem ersten Ausführen könnt ihr `deploy` verwenden.

destroy, uninstall Diese beiden Targets entsprechen dem `destroy` und `uninstall` Flag aus der letzten Übung. Falls etwas schief geht könnt ihr diese verwenden und hoffen, dass es danach wieder funktioniert 😉

Ausgaben könnt ihr auf zwei Arten machen: Ausgaben mittels Pythons `print()` Funkion werden am Ende der Ausführung des Playbooks angezeigt. Wenn ihr Dateien beschreibt könnt ihr diese in der HDFS-Übersicht sehen oder ihr nennt die Datei `result`, dann wird diese nach erfolgreicher Ausführung auf euren lokalen PC kopiert.

Aufgabe 1 Warm-up

Betrachten wir zuerst folgenden Code-Ausschnitt:

```
counter = 0
rdd = sc.parallelize(some_data)
rdd.foreach(lambda x: counter = counter + x)
print(counter)
```

a) Wieso liefert dieser Code-Ausschnitt im Zweifel ein falsches Ergebnis?

b) In welchem Fall würde der Code trotz des Fehlers das gewünschte Ergebnis liefern?

c) Wie müsste der Code-Ausschnitt angepasst werden um immer(!) das korrekte Ergebnis zu produzieren?

Aufgabe 2 1, 2, 3, 4, 5, ...

Für diese Aufgabe betrachten wir die Funktion `word_count` der Datei `python/app.py`.

a) Durch wie viele Stages wird Spark zur Ausführung dieser Funktion laufen und wodurch werden diese jeweils erzeugt?

b) Wie wird der Abstammungs-Graph von `wc3` aufgebaut sein? Welche seiner RDDs werden in welcher Stage berechnet?

c) Wie viele Tasks wird jede Stage bearbeiten? Wie kann diese Anzahl verändert werden?

d) Überprüft eure Antworten mithilfe der Spark-Übersicht.

Tipp Diese Webseite ist nur während der Laufzeit der Applikation erreichbar. Um diese künstlich in die Länge zu ziehen könnt ihr am Ende der Funktion `word_count` beispielsweise `sleep(5 * 60)` einfügen um die Daten fünf Minuten länger einsehen zu können. Nicht vergessen: `from time import sleep`

e) `text1.txt` enthält nun das komplette Buch und nicht mehr nur einen Auszug daraus (`text2.txt` bleibt unverändert). Wie ändert sich die Antwort auf Frage *a)* ?

Aufgabe 3 Zeit ist Geld

Um ein Gefühl für den Performance-Vorteil von Sparks In-Memory Processing zu bekommen, wollen wir in dieser Aufgabe die Ausführungszeiten derselben Applikation unter verschiedenen Umständen betrachten. Diese kann, neben diversen anderen Details, in der YARN-Übersicht gefunden werden.

a) Wie kann die Funktion `word_count` angepasst werden, sodass die RDDs `wc1` und `wc2` ausschließlich(!) auf den Festplatten der Worker gespeichert werden?

b) Erweitert `text1.txt` und/oder `text2.txt` um mehr Zeilen bis ein merkbarer Unterschied in der Ausführungszeit erkennbar ist.

Aufgabe 4 Talk is cheap. Show me the code

In der letzten Aufgabe wollen wir nun auch etwas programmieren. Dazu betrachten wir Trumps Tweets zwischen Mitte 2009 und Anfang 2020 aus dem Datensatz `data/trumptweets.csv`[1]. Der Datensatz ist wie folgt aufgebaut:

Link	Inhalt	Datum + Uhrzeit	Retweets	Herzen
twitter.com/...	We will build a wall!	2016-02-10 10:00:00	0	-10
...	...	...	...	...

Zur Implementierung der folgenden Aufgaben kann die, noch leere, Funktion `tweets()` aus der Datei `python/app.py` genutzt werden.

a) Zu welcher Stunde wurden die meisten Tweets abgesendet? Wie viele waren es?

b) In welchem Jahr war Trump am „*beliebtesten*“[2] auf Twitter? War dieses Jahr auch das, in dem er am meisten getweetet hat?

c) Welcher war der beliebteste Tweet während seiner Amtszeit?

[1] Rohdaten-Quelle: `https://www.thetrumparchive.com/`
[2] Um die Beliebtheit messbar zu machen addieren wir Retweets und Herzen auf. Je höher das Ergebnis, desto beliebter.