Traîtement distribué des données

Master 1 ISD

Données massives
Scala et Spark

kn@lmf.cnrs.fr

université
PARIS-SACLAY

Plan

Appache Spark

Framework de calcul distribué

- ne définie pas de stockage de donnée (peut réutiliser des bases de données, des fichiers textes, HDFS, ...)
- vient combler des lacunes de Map/Reduce
- supporte nativement plusieurs langages (Scala, Python et R)
- Est implémenté en Scala
- Fournit en plus en standard un moteur SQL et des bibliothèques de machine learning

On s'intéresse ici à l'API « core » qui se place au même niveau que Map/Reduce. Les autres composants (SQL, Streaming, Machine Learning) sont construits au dessus.

Problèmes de Map/Reduce (1)

Problèmes d'interface avec le programmeur :

- API extrèmement bas-niveau (context.write)
- Pas de sûreté de typage
- Code ultra verbeux
- Réutilisation du code difficile

Problèmes de Map/Reduce (2)

Problèmes de performances :

- Orienté disque : mauvaise utilisation de la mémoire
- Pas de partage possible de sous-tâches au niveau de l'API

Supposons une transformation qui s'exécute comme une opération M suivie de R_1 et une autre qui s'éxécute comme M suivi de R_2

Si on veut les 2 résultats, on va calculer 2 fois M

Si on ne veut pas calculer 2 fois M :

1. On calcule M et on sauve le résultat `m` dans HDFS
2. On charge `m` avec un Map identité et on envoie à R_1
3. On charge `m` avec un Map identité et on envoie à R_2
4. Si on veut combiner les résultats de R_1 et R_2 il faut refaire un map et un reduce...

Resilient Data Set

Un *Resilient Distributed Data Set* (ensemble de données distribué persistent) est une abstraction de haut niveau qui représente un calcul (et non pas son résultat) sur des données

Les RDDs sont à la base des *transformations* Spark

Le chargement des données crées un nouveau RDD (les données ne sont pas chargée, on crée juste une structure qui, quand elle sera évaluée chargera les données

On peut composer ds RDDs au moyen de *transformations*

On peut exécuter une *action* sur un RDD. Cela déclanche le calcul de toute la transformation pour obtenir un résultat final

Initialisation de Spark

Pour initialiser Spark, il faut (en toute généralité) commencer par initialiser Hadoop (si on souhaite que les données et les résultats soient stockés de façon distribuée sur Hadoop), puis démarer le service Spark. (cf la feuille de TP).

Une fois le service spark lancé, on peut écrire un fichier python comme celui ci:

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName("Exercice 1").setMaster("local")
sc = SparkContext(conf=conf)
```

La variable `sc` est le contexte Spark et va nous permettre de créer des RDD

Création d'un RDD

On peut créer un RDD de plusieurs manière :

- *`sc.emptyRDD()`*
- *`sc.parallelize(l)`* où `l` est une liste Python (ou un itérable)
- *`sc.textFile(file)`* où `file` est soit un fichier local, soit une URL de la forme `hdfs://adresse_hadoop:9000/chemin/vers/fichier`. Le fichier est lu comme un fichier texte, et renvoyé commme le RDD des lignes (type `str`)

Des fonctions plus avancées permettent aussi de créer des RDD depuis des tables SQL, des fichiers binaires, ...

Transformations de RDD

Les fonctions de transformation sont les itérateurs Scala, plus quelques nouveaux:

- `.map(f)` application de `f` pour chaque élément
- `.filter(f)`
- `.flatMap(f)` application de `f` qui renvoie une liste pour chaque élément et applatit le résultat.
- `.union(otherDataSet)`
- `.intersection(otherDataSet)`
- `.join(otherDataSet)` sur un RDD de paires (K,V) et un autre RDD de paires (K, W) renvoie les triplets (K,(V,W))
- `.distinct()`
- `.groupByKey()` (s'applique sur un RDD de paires (Clé,Valeur)
- `.reduceByKey(f)` (s'applique sur un RDD de paires (Clé,Valeur). Exécute un fold de f sur le tableau de toutes les valeurs de la même clé.
- ...

Action sur les RDDs

Les actions exécutent le RDD auquels elles sont appliquées pour renvoyer un résultat:

- `.reduce(f)`
- `.foreach(f)`
- `.collect()` retransforme le RDD en collection Python
- `.count()`
- `.take(n)`
- `.first()`

Spark en pratique

Spark est fourni avec des interpréteurs standards pour Scala, Python et R. Ce sont les interpréteurs normaux, dans lesquels sont préchargés les bibliothèques pour Spark.

Pour exécuter un programme Scala, on peut aussi l'exporter comme un jar et utiliser la commande `spark-submit` pour l'exécuter.

Démo