Bases de données documentaires et distribuées
Cours NFE04
Introduction Ãă Spark

Auteurs : Raphaël Fournier-S'niehotta, Philippe Rigaux, Nicolas Travers
prénom.nom@cnam.fr

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Qu'est-ce que Spark ?

Un **moteur d'exécution** basé sur des opérateurs de haut niveau, comme Pig.

S'appuie sur Map/Reduce (Hadoop) pour l'exécution distribuée.

Introduit un concept de collection résidente en mémoire (RDD) qui améliore considérablement certains traitements, dont ceux basés sur des itérations.

De nombreuses librairies pour la fouille de données (MLib), le traitement des graphes, le traitement de flux (*streaming*).

Les *Resilient Distributed Datasets* (RDD)

C'est le concept central : **Un RDD est une collection (pour en rester à notre vocabulaire) calculée à partir d'une source de données** (MongoDB, un flux, un autre RDD) .

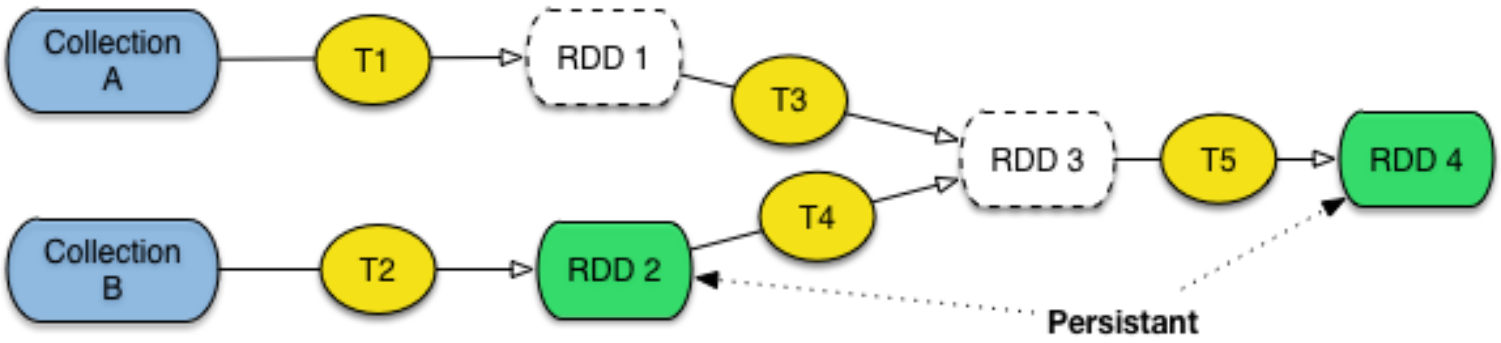
Un RDD peut être marqué comme **persistant** : il est alors placé en mémoire RAM et conservé par Spark.

Spark conserve **l'historique des opérations** qui a permis de constituer un RDD, et la reprise sur panne s'appuie sur cet historique afin de reconstituer le RDD en cas de panne.

Un RDD est un "bloc" **non modifiable**. Si nécessaire il est **entièrement recaculé**.

Un *workflow* avec RDD dans Spark

Des **transformations** (opérateurs comme dans Pig) créent des RDD à partir d'une ou deux sources de données.



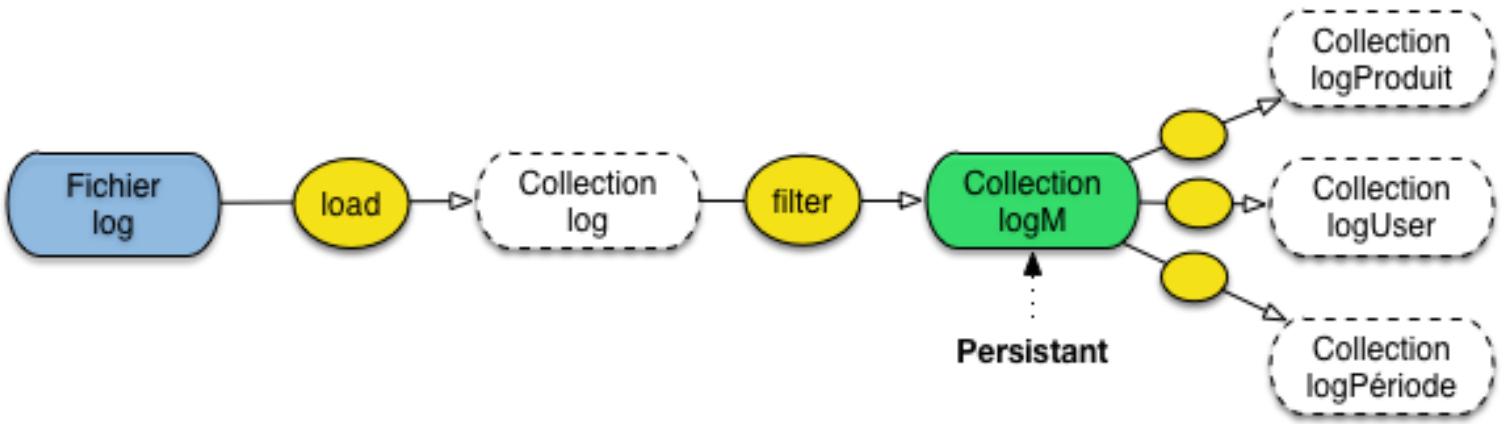
Les RDD persistants sont en préservés en mémoire RAM, et peuvent être réutilisés par plusieurs traitements.

Exemple : analyse de fichiers log

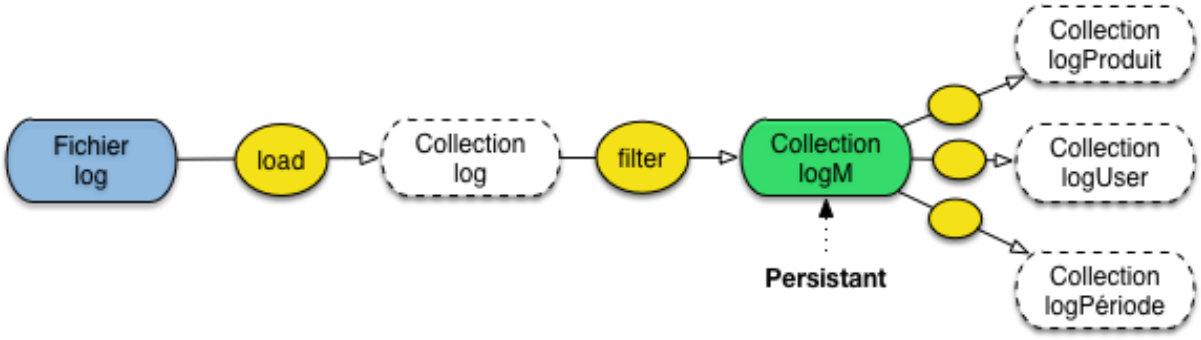
On veut analyser le fichier journal (*log*) d'une application dont un des modules (M) est suspect.

On construit un programme qui charge le *log*, ne conserve que les messsages produits par le module *M* et les analyse.

On peut analyser par produit, par utilisateur, par période, etc.



Spécification avec Spark



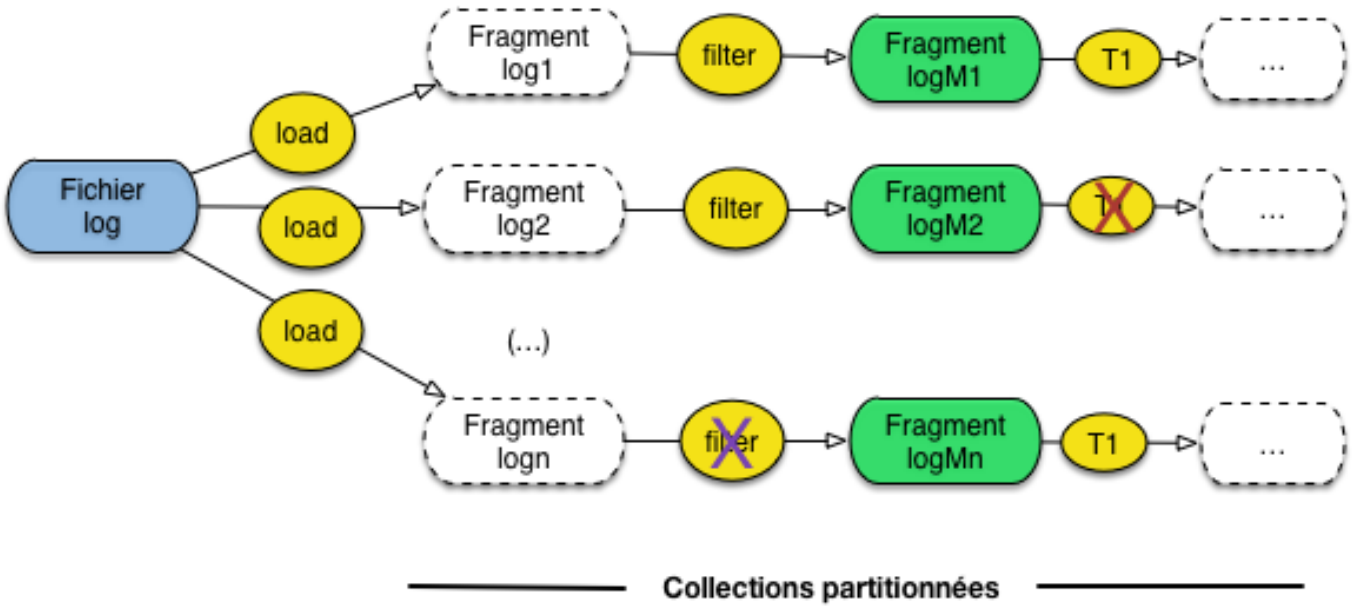

```
// Chargement de la collection
log = load ("app.log") as (...)
// Filtrage des messages du module M
logM = filter log with log.message.contains ("M")
// On rend logM persistant !
logM.persist();
```

Analyse à partir de `logM`

```
// Filtrage par produit
logProduit = filter logM with log.message.contains ("product P")
// .. analyse du contenu de logProduit
```

Reprise sur panne dans Spark

Un RDD est une collection **partitionnée**.



Si une panne affecte un calcul s'appuyant sur un fragment F de RDD persistant, il suffit de le relancer à partir de F.

Si une panne affecte un fragment de RDD persistant, Spark ré-applique la chaîne de traitement ayant constitué le RDD.

Installation

Très simple : récupérez la version courante à `http://spark.apache.org`.

Décompressez dans un répertoire `spark`.

Lancez l'intepréteur.

```
./bin/spark-shell

scala>
```

C'est un interpréteur Scala : apprenez le langage Scala !

On va implanter le compteur de termes : récupérer un fichier texte quelconque (ou `loups.txt` sur le site.)

Commandes de base

Chargement d'un fichier texte dans un RDD (un document par ligne).

```
val loupsEtMoutons = sc.textFile("loups.txt")
```

Quelques **actions** appliquées au RDD.

```
loupsEtMoutons.count() // Nombre de documents dans ce RDD
loupsEtMoutons.first() // Le premier document
loupsEtMoutons.collect() // Tous les documents du RDD
```

Les **transformations** (presque comme Pig).

```
val bergerie = loupsEtMoutons.filter(line => line.contains("bergerie"))
loupsEtMoutons.filter(line => line.contains("loup")).count() // Combinaison transf./action
```

Le compteur de mots

Comme promis, le compteur de mot Spark. D'abord on prend tous les termes (`flatMap`).

```
val termes = loupsEtMoutons.flatMap(line => line.split(" "))
```

Initialisation du comptage : chaque terme compte pour 1 (`map`).

```
val termeUnit = termes.map(word => (word, 1))
```

Je regroupe les termes, et je compte (`reduce`).

```
val compteurTermes = termeUnit.reduceByKey((a, b) => a + b)
```

Et je peux rendre **persistant** les compteurs de termes.

```
compteurTermes.persist()
```