Université de Lorraine M2 Informatique

Méthodes et programmation pour les données massives
Projet : La mine d'Alibaba
Date de rendu : 5 mars 2024 23h59

Consignes

Ce projet est à réaliser à deux (me prévenir au plus vite si vous n'avez pas de binôme). Il vous sera demandé de rendre (sur arche) une archive contenant :

1. Un rapport.
2. Le code source de l'ensemble des classes que vous avez implémentés (driver, mapper, reducer et autres composants de Hadoop).
3. Des scripts contenant les commandes HDFS que vous avez utilisés.
4. Une archive jar exécutable (et testée !).

Dans le rapport vous préciserez, pour chacune des questions posées dans ce sujet, quelles ont été les commandes exécutées (un non-expert doit pouvoir reproduire vos expériences à partir de votre archive). Vous décrirez aussi les composants que vous avez implémentés au moment où vous les utilisez (notamment la logique algorithmique sous-jacente). Vous êtes invités à avoir un retour critique sur ces derniers, notamment si vous avez repéré des limites à votre démarche.

Tous les programmes doivent être implémentés en utilisant Hadoop. En dehors de commandes HDFS, aucun autre outil ou langage n'est autorisé. Si vous utilisez une autre version de Hadoop que celle fournie par le cours, merci de le préciser et de fournir les informations sur la version et un script pour reconstruire l'archive à partir du code source.

Par défaut je testerai toutes vos applications avec l'option `-D mapreduce.job.reduces=2`. Si jamais l'un de vos jobs nécessite de n'avoir qu'un seul reducer, il faut que vous vous assuriez que ça soit bien le cas malgré les options que j'utilise et vous devrez justifier dans le rapport de pourquoi c'est nécessaire.

Vous êtes autorisés à faire des approximations, c'est à dire à renvoyer un résultat que vous savez n'être pas aussi précis qu'il pourrait l'être au vu des données d'entrée. Dans ce cas vous devrez préciser quand c'est le cas, justifier votre choix et décrire votre méthode.

L'utilisation de solutions "généralistes" (réutilisation de classes dans des étapes différentes, éviter de multiplier les classes Mapper et Reducer quand c'est possible) sera appréciée.

Un entretien d'au plus 30 minutes aura lieu avec chaque binôme le **8 mars 2024**. Il vous sera demandé de connaître le fonctionnement de l'ensemble de votre code (les réponses ne seront pas communes) et d'être capable de justifier chaque choix, notamment d'implémentation.

Note d'intention

Les questions qui suivent ne donnent pas de méthodes précises à suivre pour les résoudre, c'est volontaire et c'est à vous de trouvez une solution. Je ne répondrais donc pas ou peu aux questions traitant donc des algorithmes à implémenter. Par contre je répondrais volontiers aux questions plus techniques (comment faire une action précise en utilisant l'API Hadoop par exemple). Vous pouvez bien sûr chercher par vous même sur internet (en gardant un regard critique comme toujours) ou dans la documentation de Hadoop 3 mais n'hésitez pas à m'envoyer un email, ça peut vous faire gagner du temps (après avoir vérifié que la solution n'est pas dans les cours ou les TDs).

Par ailleurs si vous pensez qu'Hadoop n'est pas adapté pour faire ce qu'on vous demande, vous avez peut-être raison (pas systématiquement). Sur certaines questions, la structure d'Hadoop, du problème ou la taille des fichiers considérés ne justifie pas d'utiliser cet outil. Cependant le but de ce projet est avant tout de vous faire réfléchir en utilisant le modèle MapReduce et c'est pour ça que l'usage d'Hadoop est obligatoire ici.

Corpus

Les données que vous allez traiter sont un échantillon de traces provenant d'un cluster de production Alibaba (l'équivalent chinois d'Amazon), provenant du programme Alibaba Open Cluster Trace et datant de 2018. Afin de réduire la taille des données, nous ne nous intéresserons qu'à un sous-ensemble de ces traces, soit un ensemble de jobs ayant au moins une de leurs tâches actives durant une fenêtre de 2H sélectionnée au hasard (sur les environ 9 jours au total que représentent les traces fournies par le programme). Par curiosité vous pouvez jetez un oeil ici, même si toutes les informations nécessaires pour la réalisation de ce projet seront décrites dans ce sujet.

On ne fournit ici qu'un seul fichier, contenant un ensemble **d'instances**. Chaque instance fait partie d'une **tâche** qui elle-même appartient à un **job**. Pour faire une analogie, un job Mapreduce possède deux tâches (map et reduce) qui possèdent chacune au moins une instance (pour map une instance par bloc par exemple). les instances sont présentés dans un format csv contenant 11 colonnes avec les champs suivants (et dans cet ordre) :

Champs	Format
Nom de l'instance	Chaîne de caractères
Nom de la tâche à laquelle appartient l'instance	Chaîne de caractères
Nom du job auquel appartient l'instance	Chaîne de caractères
Statut de l'instance	Chaîne de caractères
Date de départ	Entier, en seconde depuis un 0 arbitraire
Date de fin	Entier, en seconde depuis un 0 arbitraire
Identifiant de la machine exécutant l'instance	Chaîne de caractères
Utilisation moyenne des CPUs	Flottant, une valeur de 100 représentant un cœur en entier
Utilisation maximale des CPUs	Flottant, une valeur de 100 représentant un cœur en entier
Utilisation moyenne de la mémoire	Flottant, normalisé entre 0 et 100
Utilisation maximale de la mémoire	Flottant, normalisé entre 0 et 100

Notons qu'il est possible que certaines instances aient des informations incomplètes ou non conformes, je n'ai volontairement pas appliqué de correctif en dehors du fait d'avoir échantillonné l'ensemble de données initial (et supprimé quelques colonnes supplémentaires que nous n'utiliserons pas).

Sauf précision contraire vous partirez systématiquement de ce fichier de base. Un extrait plus court de ce dernier vous est fourni afin de tester localement votre application.

Afin d'éviter d'avoir beaucoup de blocs, ce qui dégraderait les performances de vos applications, n'oubliez pas de changer dans le fichier `hdfs-site.xml` le champs `dfs.block.size` (vous pouvez commentez les lignes correspondantes).

Nomenclature des tâches

Si les jobs et les instances possèdent a priori un nom unique, vous pouvez remarquez que certaines tâches provenant de jobs différents possèdent le même nom. La raison est que le nom de chaque tâche est interne au job auquel elle appartient. On trouve deux types de noms. Si ce dernier est de la forme `task_XXXXX` (`XXXXX` étant ici une suite de caractères aléatoires), alors la tâche est indépendante. Sinon le nom de la tâche est composé d'une lettre et d'un nombre, suivi d'éventuellement plusieurs autres nombres séparés par des `"_"`, ce qui peut donner par exemple `M1`, `J4_2` ou `R7_5_3`. La lettre n'aura ici pas d'importance, par contre le premier nombre sert à connaître le numéro propre à la tâche et les nombres suivants nous permettent

de connaître ses dépendances (ces tâches font partie d'un job qu'on peut représenter sous forme de DAG, un graphe orienté acyclique). Par exemple la tâche `J4_2` dépend de la tâche numéro `2` et `R7_5_3` dépend de la tâche `5` et de la tâche `3`.

Étape 1 — Jobs et tâches

Dans un premier temps on cherchera à obtenir quelques informations simples sur les jobs et les tâches représentées par ces instances. Nous cherchons à produire deux documents :

- Dans l'un une liste des tâches présentes et pour chacune (sur la même ligne) : la durée de la tâche, son nombre d'instances et son nombre de "stragglers" [1], c'est à dire le nombre d'instances durant au moins 20% de temps en plus que la moyenne des autres instances.
- Dans l'autre une liste des jobs présents et pour chacun (sur la même ligne) : leur nombre de tâches et leur nombre d'instances.

Étape 2 — Pic de consommation

On s'intéresse ici à la consommation de cet ensemble de tâches. Plus particulièrement on s'intéresse à la quantité de cœurs utilisés chaque seconde et on cherchera à produire une liste de tous les pics de consommations, triée par ordre décroissant d'importance. Plus précisément notre liste aura trois colonnes : l'une comprenant la seconde auquel ce pic a lieu, la seconde le nombre de cœurs utilisés lors de ce pic et la troisième la durée de ce pic (c'est à dire le nombre de secondes où cette consommation se maintient). Le tri se fera en fonction de la seconde colonne. On définit un pic comme un instant où :

- Le nombre de cœurs utilisés est strictement plus grand que la seconde d'avant.
- Le nombre de cœurs utilisés diminue strictement à la fin du pic.

Étape 3 — Estimation de la puissance de chaque machine

On se propose ici d'estimer le nombre de cœurs dont dispose chaque machine. Pour cela on se basera sur leurs pic de consommation respectifs (qui permettent a minima d'obtenir une borne inférieur du nombre réel de cœurs). Produisez une liste des machines avec pour chacune leur capacité estimée en nombre de cœurs.

1. Littéralement "trainards" en français mais le terme n'est pas canonique.

Étape 4 — Estimation des capacités mémoire de chaque machine, partie 1

Les capacités mémoires étant normalisées il ne nous est donc pas possible d'avoir une mesure précise de celle-ci pour chaque machine. On va cependant essayer de les hiérarchiser. On va faire l'hypothèse suivante : les instances d'une même tâche possèdent les mêmes caractéristiques, notamment dans leur utilisation mémoire. Dès lors si deux machines exécutent deux instances d'une même tâche et n'utilisent pas la même mémoire, alors l'une d'elle possède plus de mémoire que l'autre.

Nous allons donc dans un premier temps produire un graphe donnant pour chaque couple de machines le ratio estimé entre les mémoires de chaque machine (si bien sûr on peut estimer un tel ratio). Vous expliciterez la représentation de votre graphe dans votre rapport.

Étape 5 — Un peu d'options

Modifiez votre travail de l'étape 1 de façon à pouvoir refaire les calcul en :

- Ne considérant pas certaines instances selon leur statut.
- Classant les tâches et les jobs selon la colonne de son choix.
- Séparant les tâches en plusieurs fichiers finaux, chacun correspondant à un intervalle de durée, de nombre d'instance ou de nombre de stragglers (par exemple un fichier avec les tâches de moins de 3s, un autre avec celles de 4 à 10s et un dernier avec les tâches de 11s et plus). La colonne discriminante est bien sûr au choix de l'utilisateur.

Vous préciserez dans le rapport comment utiliser précisément votre application. Le lancement de votre application sans aucune option doit correspondre à l'exécution de votre étape 1.

Étape 6 — Estimation des capacités mémoire de chaque machine, partie 2

Si vraiment vous avez le temps (et fini le reste), essayez de transformer le graphe de l'étape 4 en une liste de machines, chacune associée à une valeur. Cette dernière devra représenter la quantité de mémoire de cette machine comparativement à celle de référence (a qui on aura attribué le nombre 100).