Cahier de laboratoire

Projet de construction du make distribué

Anas Chakir
Abdelhadi Nasmane
Adnane Elasli
Ayman Lmimouni

December 16, 2024

Contents

1 Introduction

Ce document présente les expérimentations menées dans le cadre du projet, en décrivant de manière détaillée les étapes nécessaires pour reproduire les tests. Les sections suivantes fournissent des informations sur les versions de code utilisées, les configurations des machines, les scripts d'exécution ainsi que les fichiers de résultats obtenus. Cette organisation permet d'assurer la transparence et la reproductibilité des résultats.

2 Installation de Spark et Java

Pour utiliser Spark sur un système distribué, il est nécessaire d'installer Spark ainsi qu'une version de Java compatible sur chaque site. Un script d'installation est fourni dans le répertoire `systemes-distribues/scripts/install_spark` pour automatiser cette tâche (`systemes_distribues` est le nom du répertoire créé après que vous clônez notre Depot Git).

Dans ce répertoire, vous trouverez le script `setup_nodes.sh`, qui doit être copié dans votre répertoire personnel sur une des machines du site `~/` et exécuté. Une fois l'exécution terminée, il est important de copier le fichier `.bashrc` généré dans votre répertoire personnel sur site `~/`. Si vous possédez déjà un fichier `.bashrc`, vous devez intégrer le contenu du nouveau fichier à celui existant pour garantir la prise en charge de Spark et de Java sur votre système.

3 Lancement du Make Distribué

Pour lancer notre make distribué, vous devez utiliser les scripts disponibles dans le répertoire `systemes-distribues/scripts/setup_cluster`. Voici les étapes à suivre :

1. Déplacez-vous vers le répertoire `systemes-distribues/scripts/setup_cluster`.
2. Allouez les machines que vous souhaitez utiliser en renseignant leurs sites et leurs noms dans le fichier `config.txt`. Vous allez avoir besoin de l'adresse IP du master node, exécutez hostname -I sur cette machine pour la récupérer.
3. Copiez le même contenu de `config.txt` dans le fichier `kill_machines.txt`.
4. Le dernier argument de la commande sera NONE si vous voulez exécuter un test qui existe deja dans le repo, sinon vous devez fournir le chemin vers un repertoire local **testX** de votre choix, contenant un Makefile et ses fichiers initiaux.

3.1 Commandes pour Lancer le Script

Les commandes pour lancer le script `setup_spark_clusters.sh` varient selon les configurations souhaitées :

Mode NFS sur les machines d'un même site : Les machines partagent un répertoire commun dans votre `~/`. La commande est :

```
./setup_spark_clusters.sh config.txt /home/{username}/systemes-distribues/src/
    test/resources/{testX}/Makefile /home/{username}/systemes-distribues /home/{
    username}/spark-3.5.3-bin-hadoop3 {target_to_execute} {username} NFS NO_TMP
    {path_to_local_makefile_dir|NONE}
```

Exemple: Sur votre machine personnelle, vous avez un répertoire `~/Desktop/testMamaMiaMarchello`, qui contient un `Makefile` et ses fichiers initiaux, et vous voulez exécuter la tâche `all`. Pour cela, vous devrez :

1. Vous déplacer vers `systemes-distribues/scripts/setup_cluster`.

2. Remplir le fichier `config.txt`.

3. Exécuter la commande suivante :

```
./setup_spark_clusters.sh config.txt /home/gmounier/systemes-distribues/src/
    test/resources/testMamaMiaMarchello/Makefile /home/gmounier/systemes-
    distribues /home/gmounier/spark-3.5.3-bin-hadoop3 all gmounier NFS NO_TMP ~/
    Desktop/testMamaMiaMarchello
```

Mode NO_NFS avec stockage temporaire (`/tmp`) : Pour forcer l'exécution dans `/tmp`, utilisez la commande suivante :

```
./setup_spark_clusters.sh config.txt /tmp/systemes-distribues/src/test/
    resources/{testX}/Makefile /tmp/systemes-distribues /tmp/spark-3.5.3-bin-
    hadoop3 {target_to_execute} {username} NFS TMP {path_to_local_makefile_dir|
    NONE}
```

Mode hybrid (NFS entre machines d'un site, NO_NFS entre deux machines de différents sites : utilisez :

```
./setup_spark_clusters.sh config.txt /home/{username}/systemes-distribues/src/
    test/resources/{testX}/Makefile /home/{username}/systemes-distribues /home/{
    username}/spark-3.5.3-bin-hadoop3 {target_to_execute} {username} NO_NFS
    NO_TMP {path_to_local_makefile_dir|NONE}
```

3.2 Prérequis

Avant d'exécuter ces commandes, assurez-vous que Spark et Java ont été correctement installés sur toutes les machines impliquées.

4 Tests et résultats

Nous avons effectué plusieurs tests sur différentes versions de notre Make. Cependant, nous avons décidé de présenter uniquement les tests les plus significatifs et représentatifs : `test6`, `test7`, `test8`, `test9`, nommés dans le répertoire `src/test/resources/testX`.

4.1 Informations sur les tests

Date des tests : Les tests et leurs résultats ont été réalisés pendant la période du 11 décembre au 16 décembre.

Machines utilisées : Les tests ont été exécutés sur 14 machines `paradoxe` situées sur le même site à Rennes, chacune disposant de 104 cœurs.

Version du git : La plupart des tests ont été réalisés avec les versions les plus récentes du dépôt git. Les versions précédentes présentaient peu de différences, à part quelques optimisations mineures. (Veuillez utiliser la dernière version pour tester)

4.2 Scripts utilisés pour les tests

Dans le répertoire `src/test/resources/`, plusieurs répertoires de tests sont disponibles, y compris ceux fournis par le professeur. Chaque répertoire contient un `Makefile` et ses fichiers initiaux ou un script pour générer un `Makefile`. La reproductibilité des résultats est garantie grâce à l'utilisation de `random.seed(42)` dans les scripts exploitant l'aléa.

Les expérimentations sont organisées en deux répertoires principaux :

- `make-j_VS_make-nfs`
- `make-j_VS_make-no-nfs`

Chacun de ces répertoires contient des sous-répertoires dédiés (du même nom) pour les `output_logs_testX`, qui stockent les résultats pour le `testX` exécuté avec des scripts de test spécifiques.

Script principal : Le script `setup_spark_clusters.sh` est utilisé pour lancer les calculs de chaque test. Il s'appuie sur les scripts `clean_after_iteration`, `run_after_iteration`, et `stop-all-spark` pour optimiser les itérations. Pour chaque test (dans le répertoire `output_logs_testX`), le script crée Y sous-répertoires nommés `with_Y_machines` (`Y` allant de 1 à 14 dans notre cas), où sont stockés :

- Les fichiers de logs `log_iteration_Y_Z.txt` (`Z` allant de 1 à 4).
- Les fichiers de résultats bruts `log_iteration_Y_Z` pour le `testX`, exécuté sur `Y` machines avec `Z` répétitions (4 dans notre cas).

Le script génère également un fichier `configY.txt` pour déployer sur les machines nécessaires.

Chaque répertoire `output_logs_testX` contient aussi `Z` fichiers `seq_file_time_Z.txt` enregistrant les performances de `make -j` (sur une seule machine), qui servent à tracer le modèle parfait du meilleur cas.

4.3 Génération des graphes

Les deux répertoires principaux contiennent des scripts Python pour tracer les graphes des performances en s'appuyant sur les fichiers de performance d'un `testX` donné. Ces graphes incluent :

- Performances
- Efficacité
- Accélération

Les graphes générés sont sauvegardés dans le répertoire `output_logs_testX` correspondant.

Pour générer des graphes comparant les performances pour un même `testX` sur les versions `NFS`, `no NFS`, et `make-j`, deux scripts Python situés dans le répertoire parent `scripts` sont utilisés :

- `efficiency_acceleration_combined.py`
- `plot_execution_times_all.py`

Ces scripts prennent le `testX` comme argument en ligne de commande et sauvegardent les graphes générés dans le répertoire `scripts` où ils sont exécutés.

4.4 Les répertoires `graph_build_estimation` et `spark_losses_estimation`

Ces deux répertoires contiennent les scripts permettant de modéliser nos différentes versions de `make`.

`graph_build_estimation` Ce répertoire contient une version modifiée de la fonction `Main.java`, utilisée pour générer le graphe de dépendances et éviter l'éxecution entière du Makefile (on s'interesse uniquement à la construction du graphe). Celle-ci s'appuie sur les fichiers `MakefileParser.java` et `TaskGraph.java`.

Pour cela, nous avons utilisé un ensemble de *Makefiles* construits de la manière suivante :

- Le script `generateAllToAllTree.py` génère les *Makefiles* en fonction de paramètres définis (nombre de niveaux et nombre de tâches par niveau dans une plage spécifiée en ligne de commande).
- Le script `graph_theorique_perf.py` entraîne un modèle de régression linéaire sur ces *Makefiles*.
- Les coefficients du modèle sont enregistrés dans un fichier `model_coefficients.txt`.

Ces coefficients permettent d'estimer les performances théoriques du modèle. Ce dernier a été utilisé par le script d'affichage pour générer le graphe de performance théorique, correspondant notamment aux tests 8 et 9.

`spark_losses_estimation` Le répertoire `spark_losses_estimation` a un objectif similaire : prédire le temps nécessaire à `Spark` pour distribuer les tâches, diffuser (*broadcast*) les variables nécessaires et gérer les connexions entre le *driver* et les *workers*.

Cependant, contrairement au premier répertoire, celui-ci ne contient pas de fichiers Java séparés. À la place, il inclut un fichier `.jar` qui regroupe l'intégralité de `make`. Cette étape est nécessaire car nous devons désormais exécuter notre version de `make` pour collecter les mesures requises.