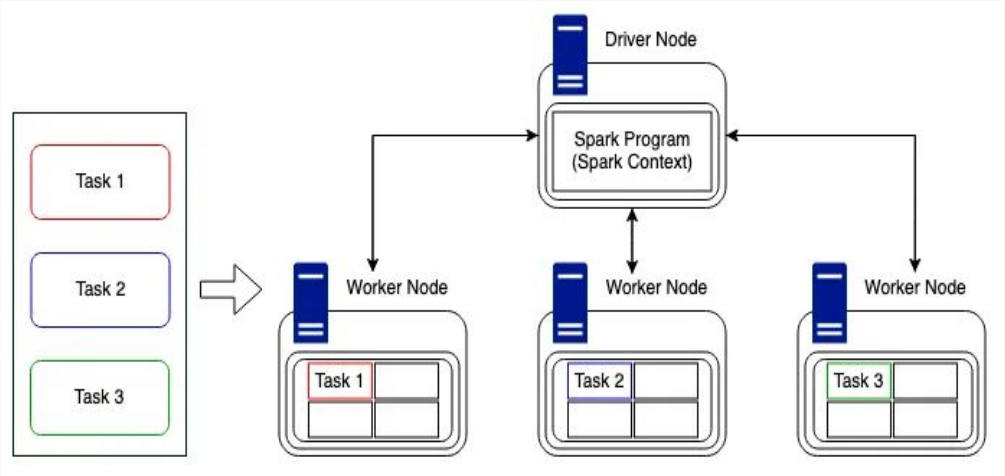
Marke un make GNU distribué (construit par Java et Apache Spark)

Anas Chakir | Abdelhadi Nasmane | Adnane Elasli | Ayman Lmimouni

Le dépôt Git du projet: https://gitlab.ensimag.fr/chakiran/systemes-distribues

- ★ Sockets pour la communication
- ★ Programmation concurrente
- ★ Gestion des fichiers (java.nio)
- ★ Compatibilité avec Spark
- ★ Portabilité multiplateforme

- ★ Architecture Master-Worker
- ★ Performance avec RDD
- ★ Tolérance aux pannes
- ★ Passage à l'échelle horizontal



- ★ Le master exécute la dernière tâche (on peut avoir un worker sur le même noeud)
- ★ Les noeuds peuvent appartenir à différents sites
- ★ Version multi-coeurs et localité des fichiers

Choix d'implantation:

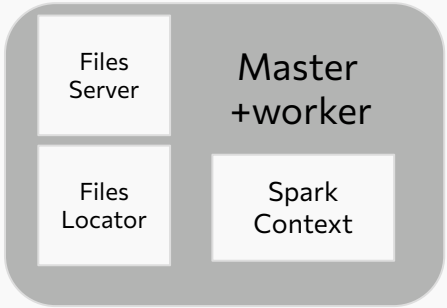
Version NFS:

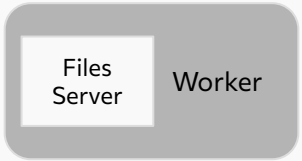
- ★ Construction d'un graphe de dépendances (Kahn's algorithm) pour être compatible avec les RDDs de Spark.
- ★ Exécution des tâches par niveau sur les différents coeurs des différentes machines.
- ★ Les tâches d'un niveau sont lancées uniquement après la fin de toutes les tâches du niveau précédent.
- ★ Arrêt d'exécution des prochains niveaux si une des tâches du niveau précédent échoue.
- ★ Déploiement automatique: scripts, fichier de configuration...

Version Sans NFS:

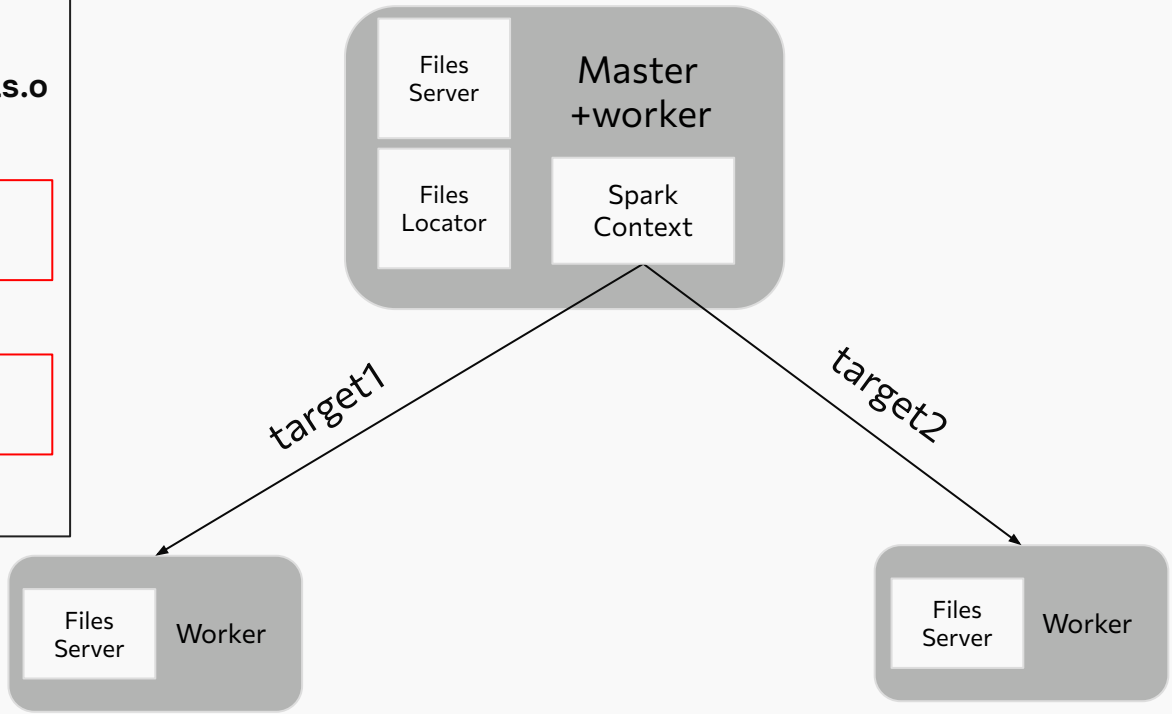
- ★ Calcul de dépendances complet (tâche -> toutes ses dépendances directes et indirectes) : Algorithme Parallèl.
- ★ Sur chaque machine (et pas coeur) on lance un serveur ServeFile qui s'occupe de l'envoi des fichiers générés.
- ★ Le serveur FileLocator sur le master reçoit et stocke Map<tâche, noms de fichiers générés> des workers, et transmet Map<Adresse IP, noms de fichiers générées> comme réponse à List<dépendances>.
- ★ Chaque coeur crée un répertoire temporaire pour exécuter sa tâche pour éviter les problèmes de concurrence (lecture, ecriture et copie de fichiers au même temps).

```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

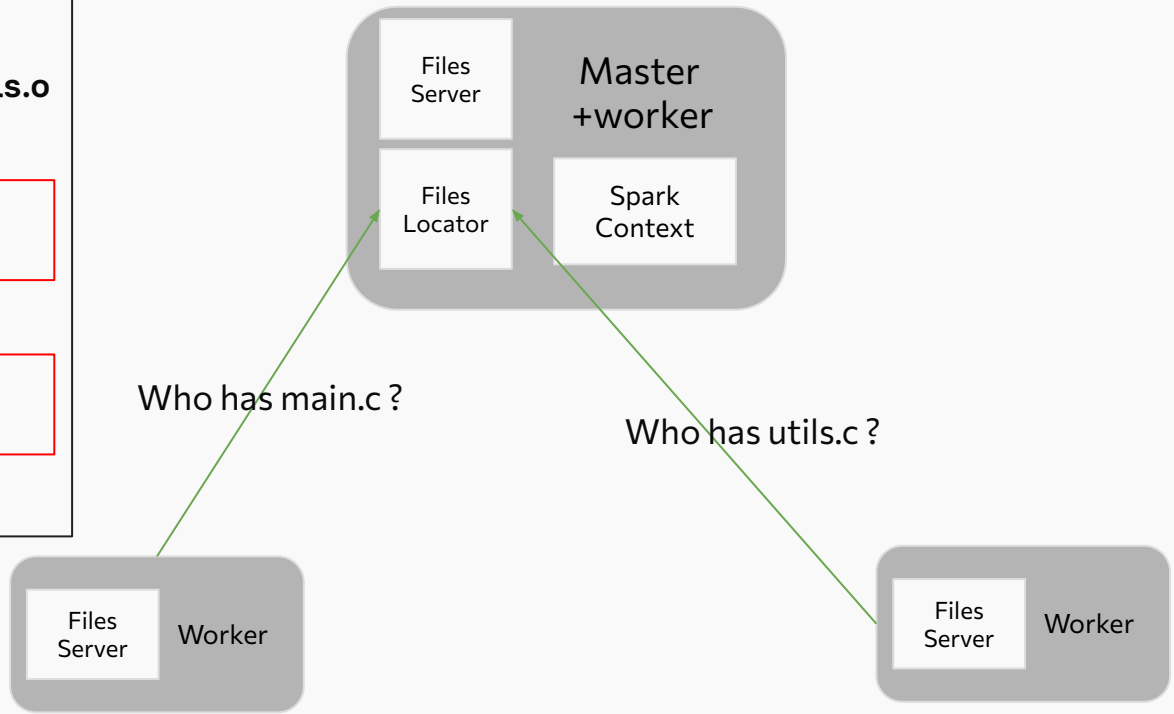




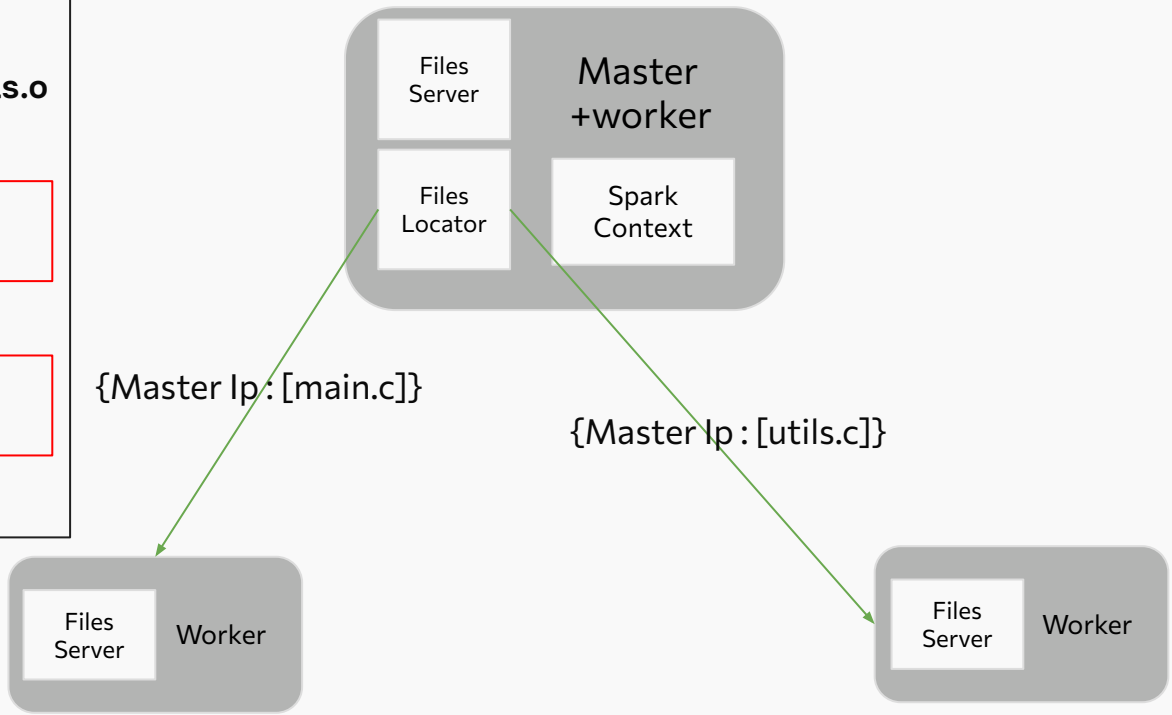
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```



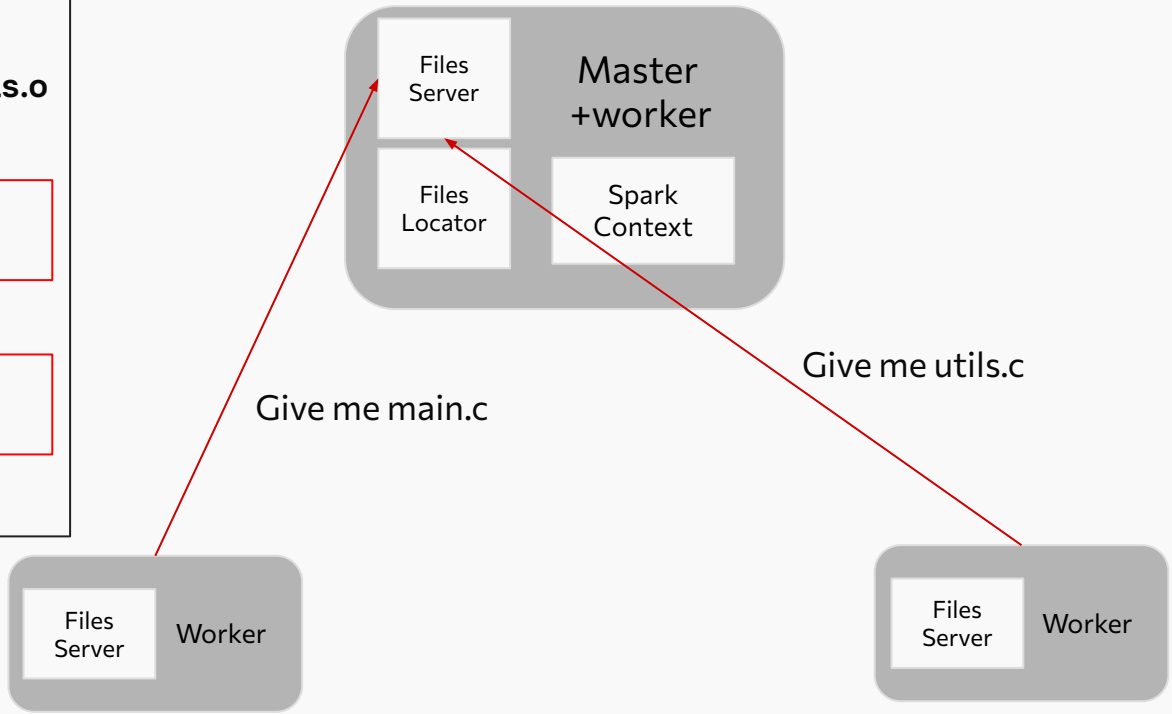
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```



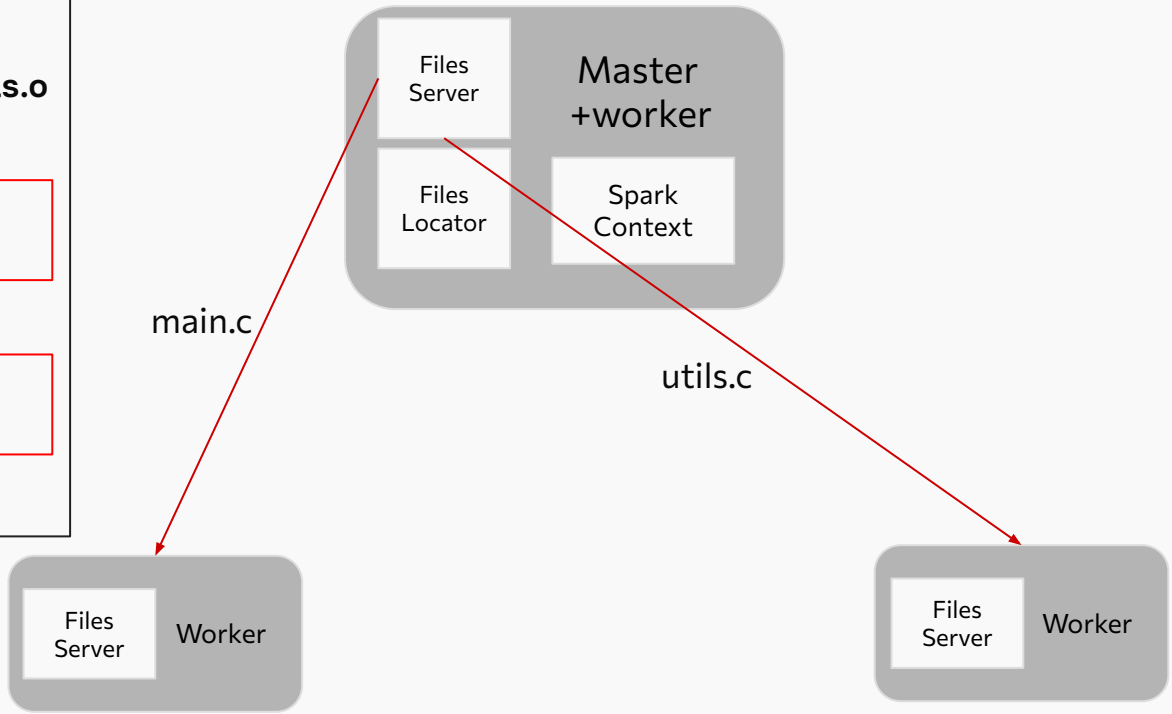
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```



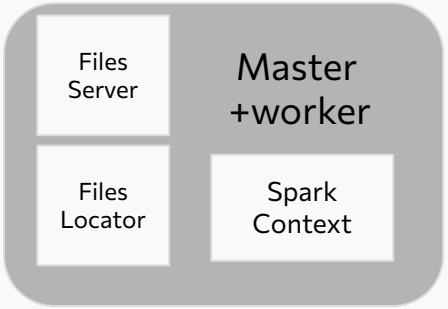
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

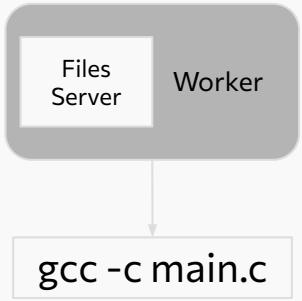


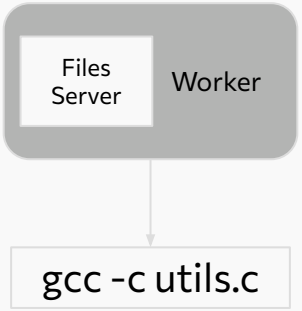
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```



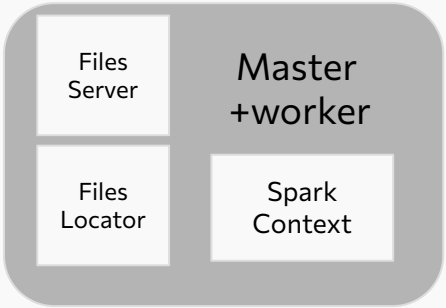
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

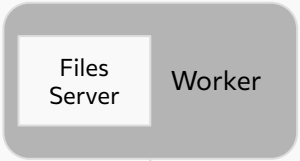


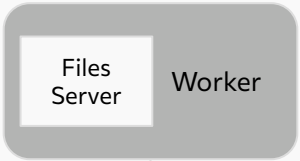


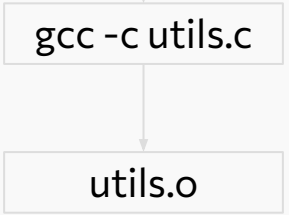


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

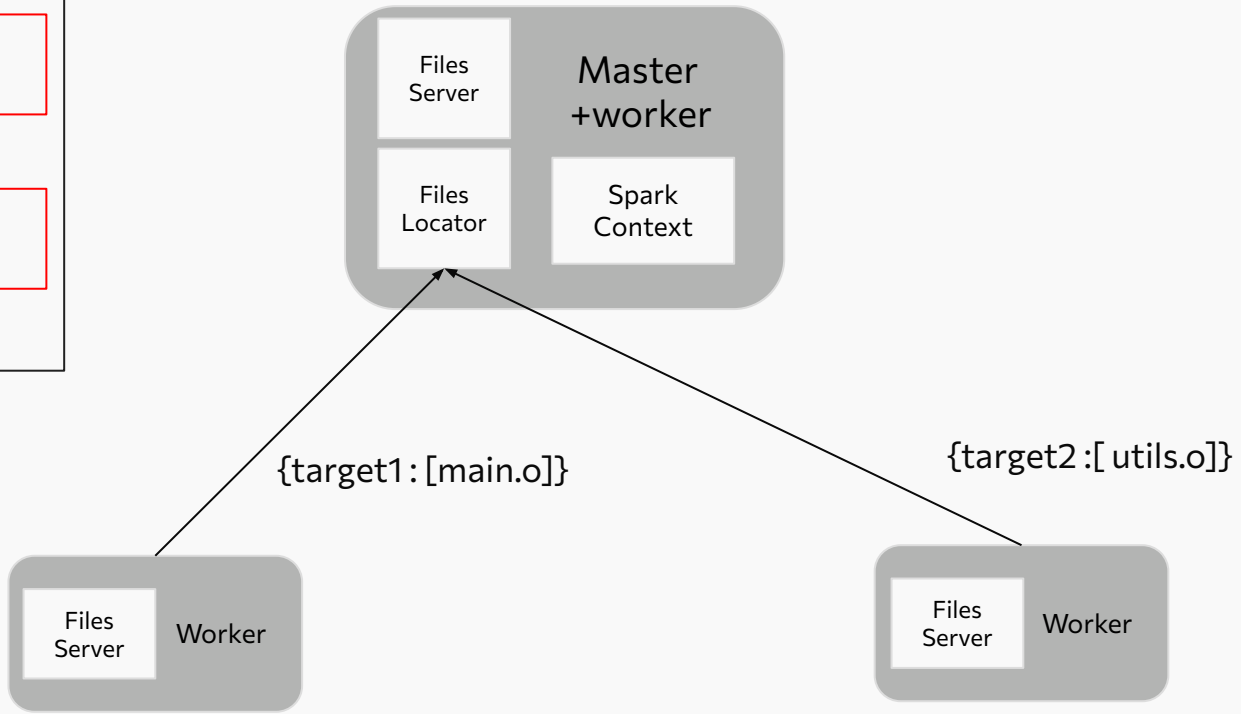






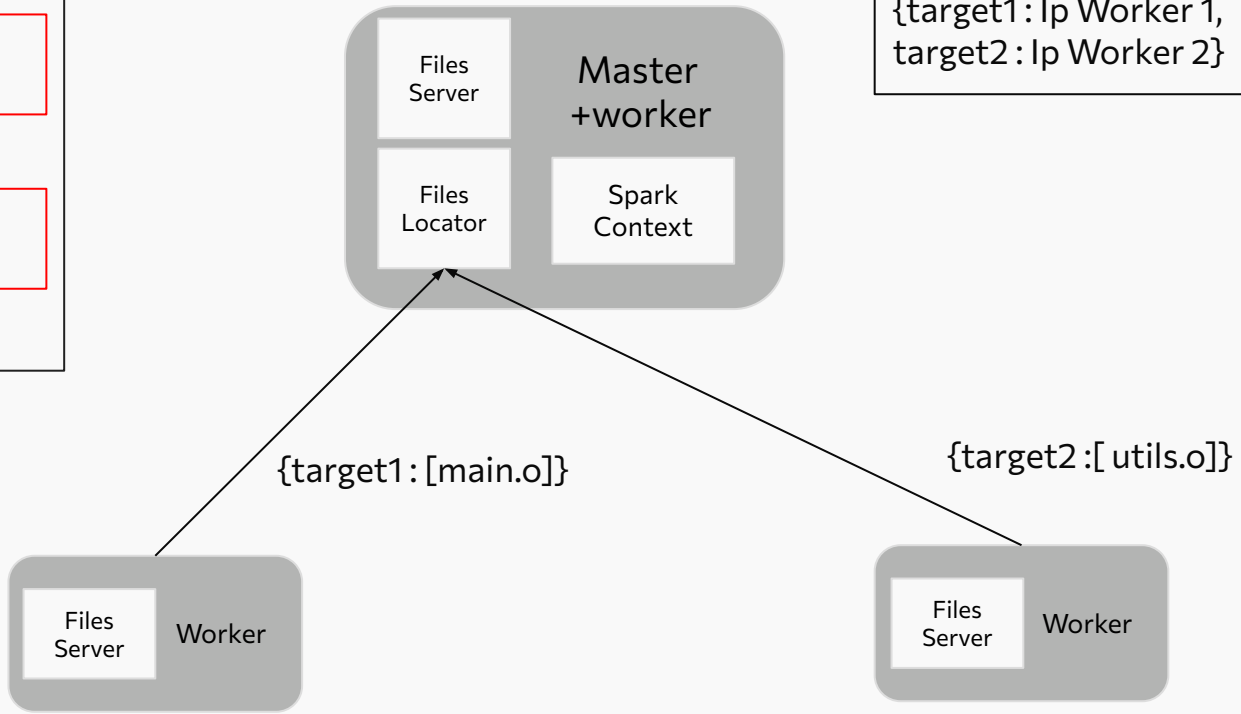


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

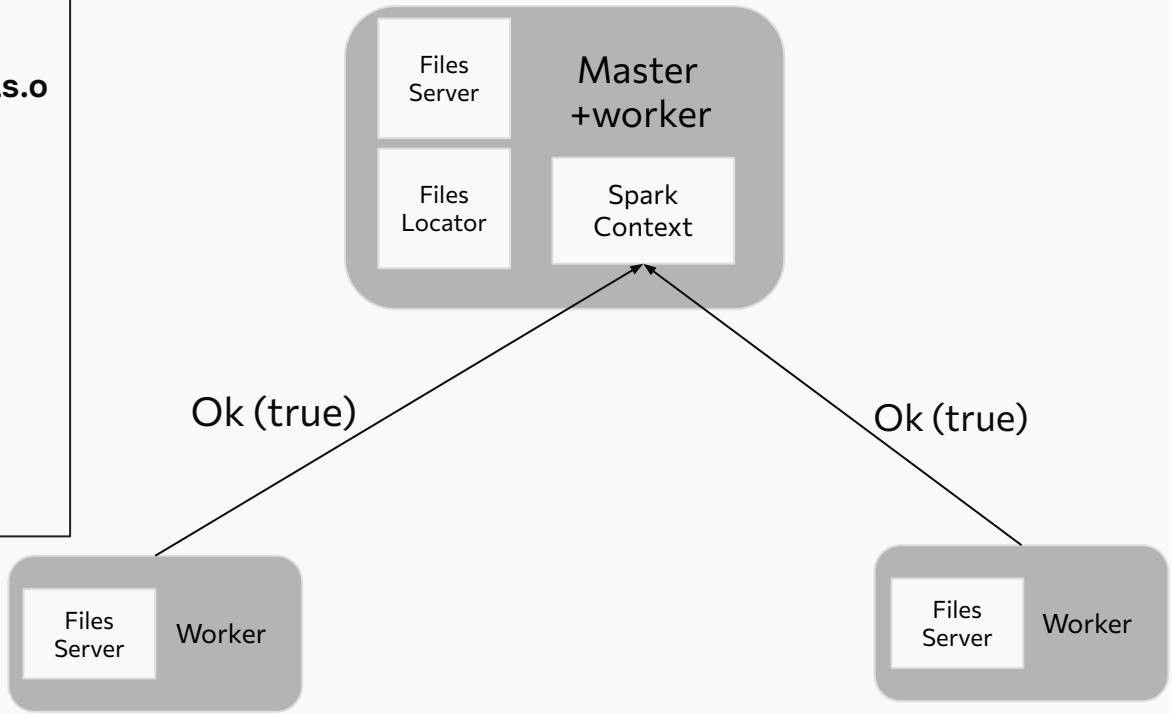


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

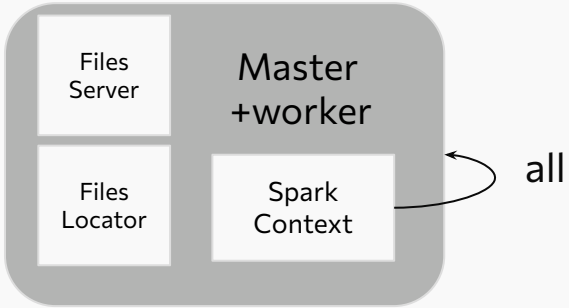
FileLocator stores also:
{target1 : Ip Worker 1,
target2 : Ip Worker 2}

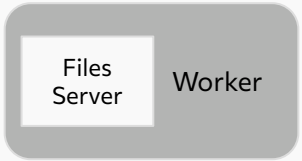


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

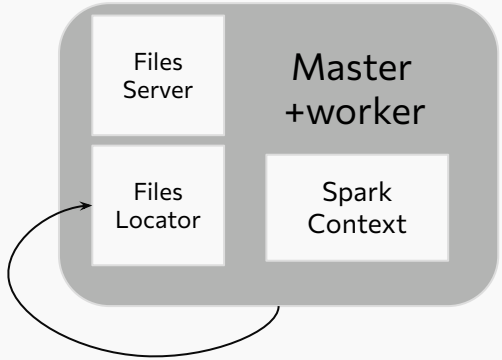


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

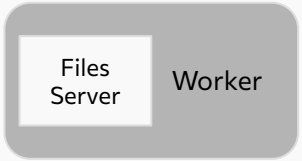


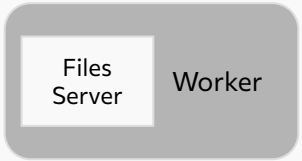


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

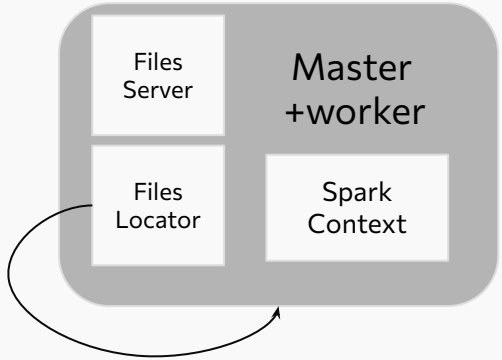


Who has [target1, target2] ?

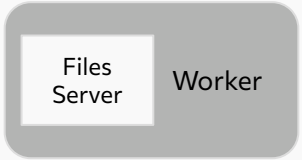




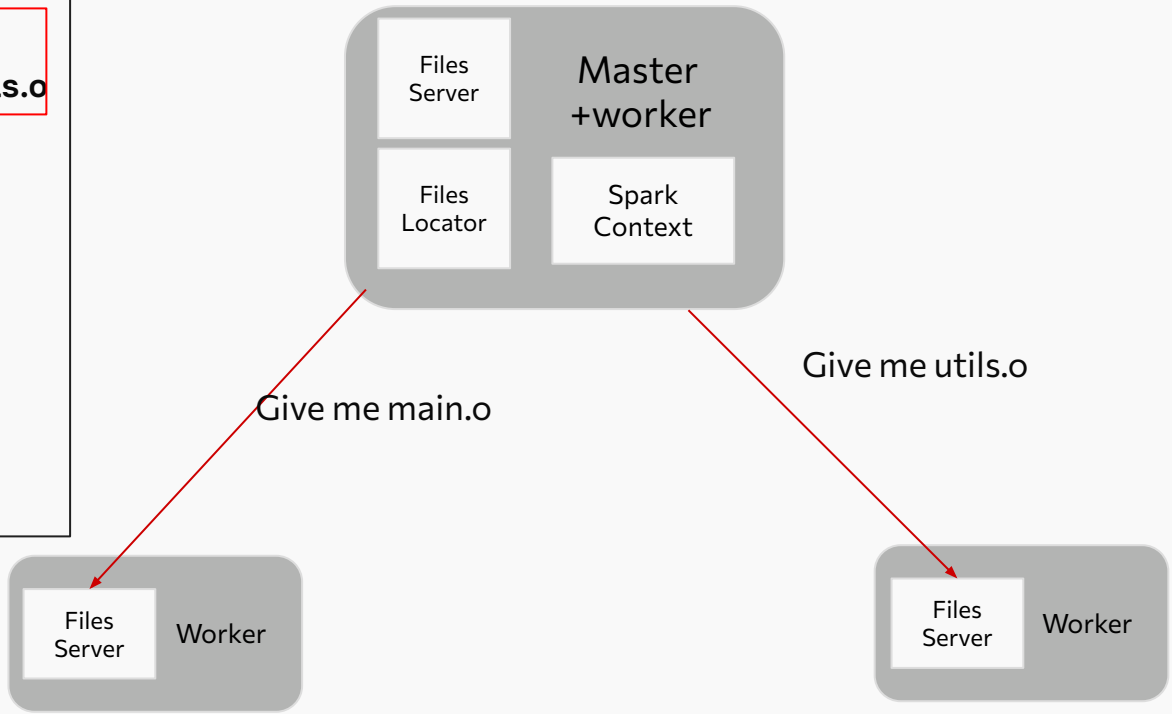
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```



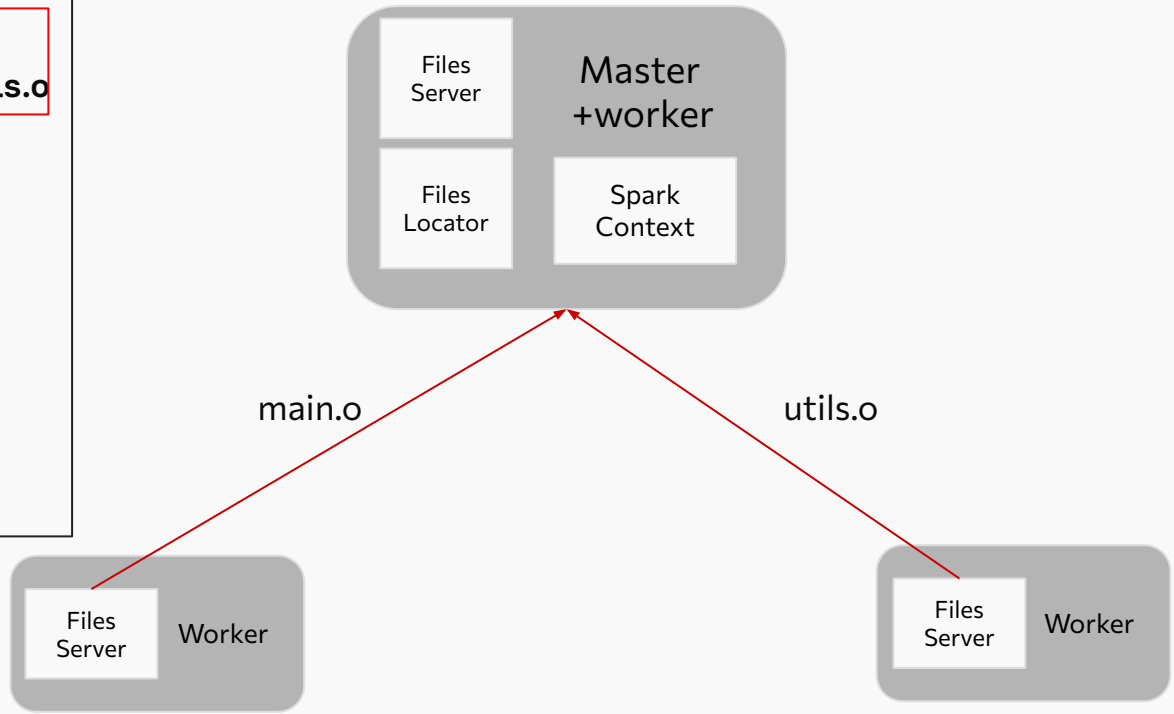
{Ip Worker 1 : [main.o], Ip Worker 2 : [utils.o]}

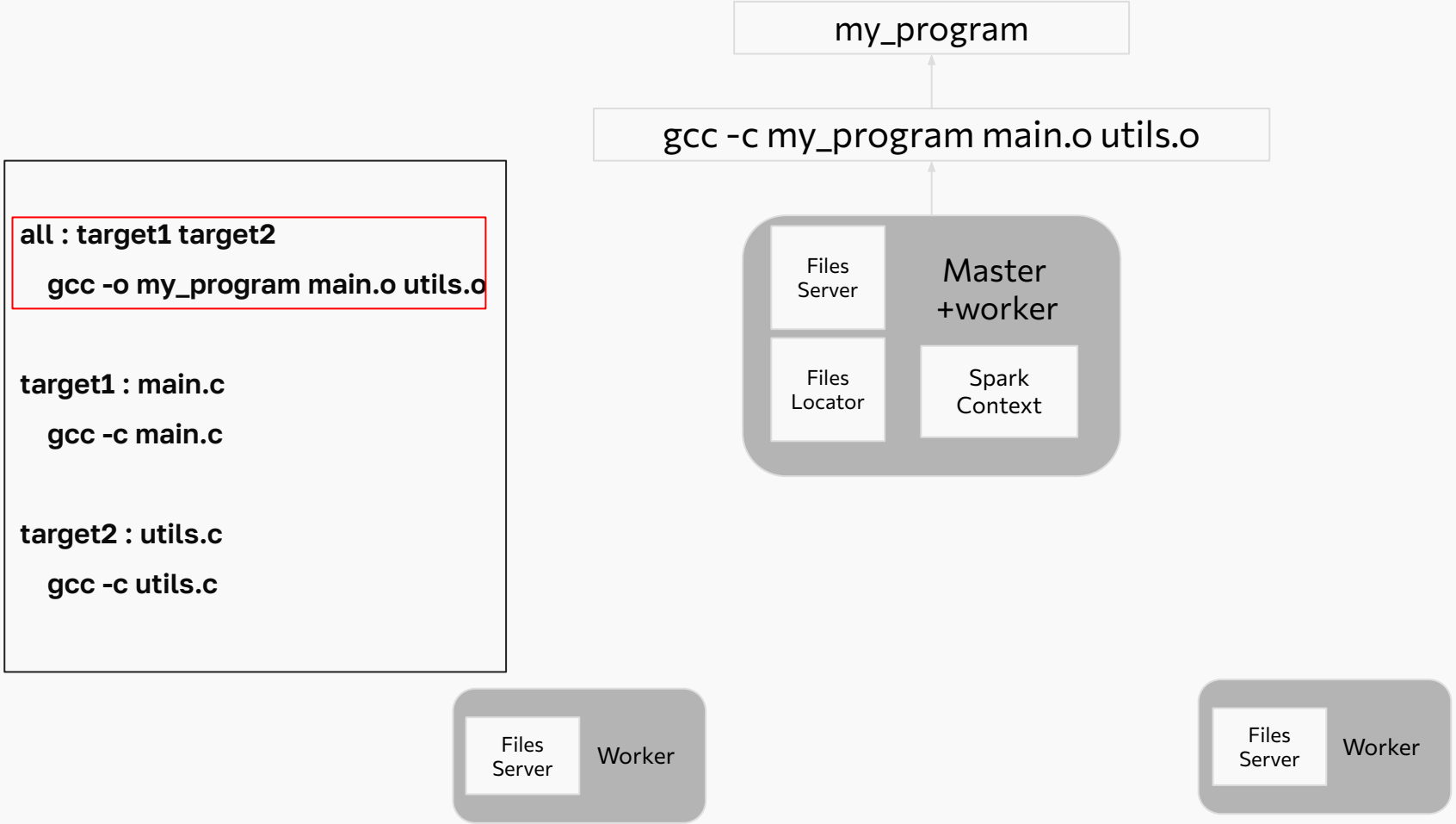


```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

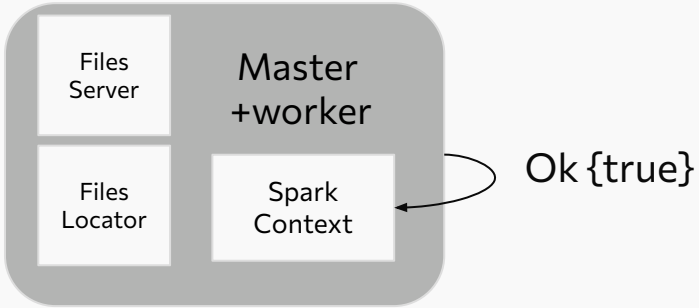


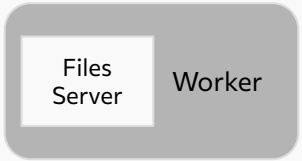
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```



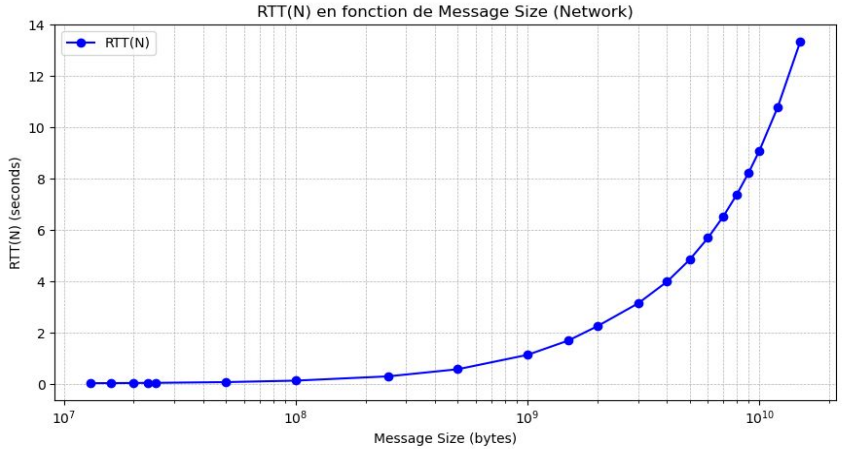

my_program
gcc -c my_program main.o utils.o
all : target1 target2
gcc -o my_program main.o utils.o
target1 : main.c
gcc -c main.c
target2 : utils.c
gcc -c utils.c
Files Server
Master +worker
Files Locator
Spark Context
Files Server
Worker
Files Server
Worker

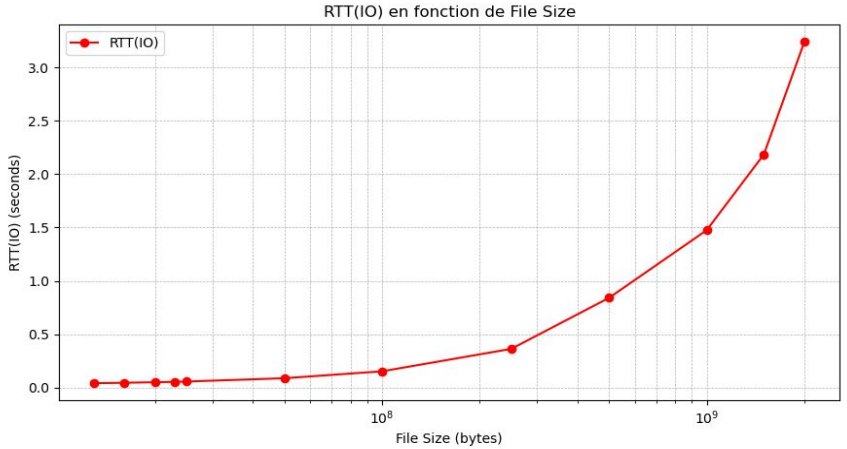
```
all : target1 target2
    gcc -o my_program main.o utils.o

target1 : main.c
    gcc -c main.c

target2 : utils.c
    gcc -c utils.c
```

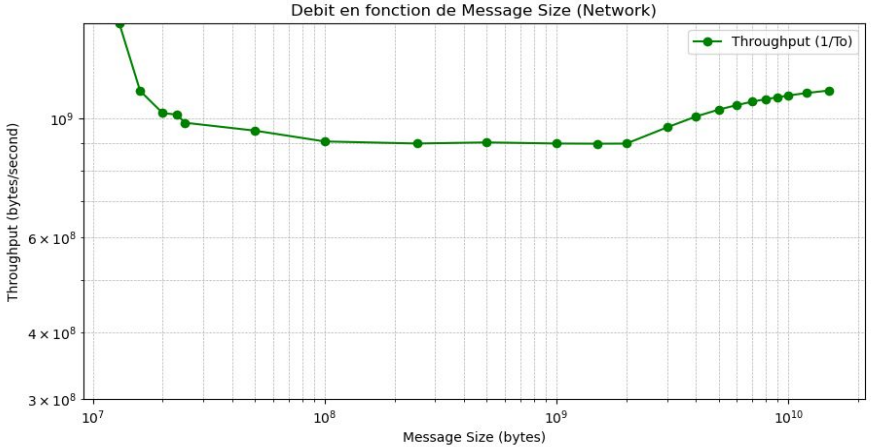


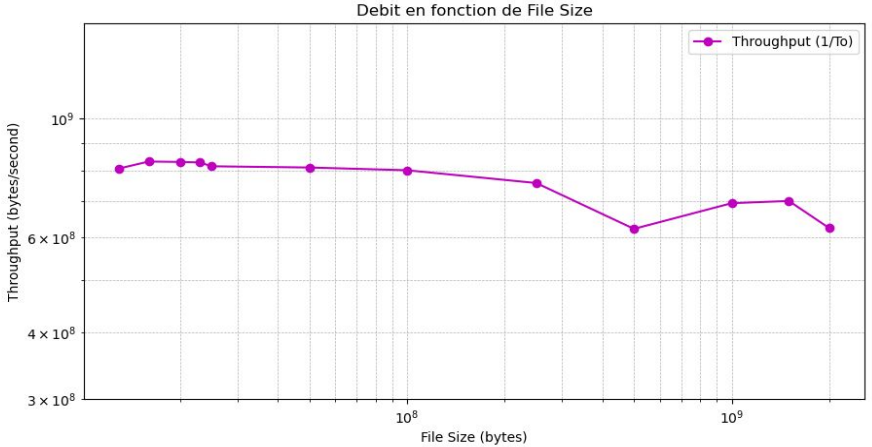


Les performances du réseau entre deux machines (Rennes) chacune de debit 1.25 GByte/secondes









Modèle de Marke

Soit m le nombre total de cœurs, l le nombre de niveaux, $N = [n_1, n_2, \ldots, n_l]$ les nombres de tâches dans chaque niveau, T les temps que prend les commandes, F les temps de transfert des fichiers et G les temps pour contacter FileLocator.

$$T = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1n_1} \\ t_{21} & t_{22} & \ldots & t_{2n_2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{l1} & t_{l2} & \ldots & t_{ln_l} \end{bmatrix} \quad F = \begin{bmatrix} f_{11} & f_{12} & \ldots & f_{1n_1} \\ f_{21} & f_{22} & \ldots & f_{2n_2} \\ \vdots & \vdots & \ddots & \vdots \\ f_{l1} & f_{l2} & \ldots & f_{ln_l} \end{bmatrix} \quad G = \begin{bmatrix} g_{11} & g_{12} & \ldots & g_{1n_1} \\ g_{21} & g_{22} & \ldots & g_{2n_2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{l1} & g_{l2} & \ldots & g_{ln_l} \end{bmatrix}$$

Alors le modèle au pire des cas, où chaque tâche d'un niveau i dépend de toutes les tâches du niveau $i-1$ (et donc dépend indirectement de toutes les tâches des niveaux $j < i$), s'écrit comme suit :

Formule générale

$$T_{\text{total}} = T_{\text{parse}}(N) + T_{\text{graph}}(N) + T_{\text{sparkConfig}} + \sum_{i=1}^{l} \left(\frac{\sum T[i] + \sum F[i] + \sum G[i]}{m} + \max(T[i] + F[i] + G[i]) \right) + T_{\text{sparkOverhead}}$$

NFS

$$T_{\text{total}} = T_{\text{sparkConfig}} + \sum_{i=1}^{l} \left(\frac{\sum T[i]}{m} + \max(T[i]) \right) + T_{\text{sparkOverhead}}$$

Sans NFS

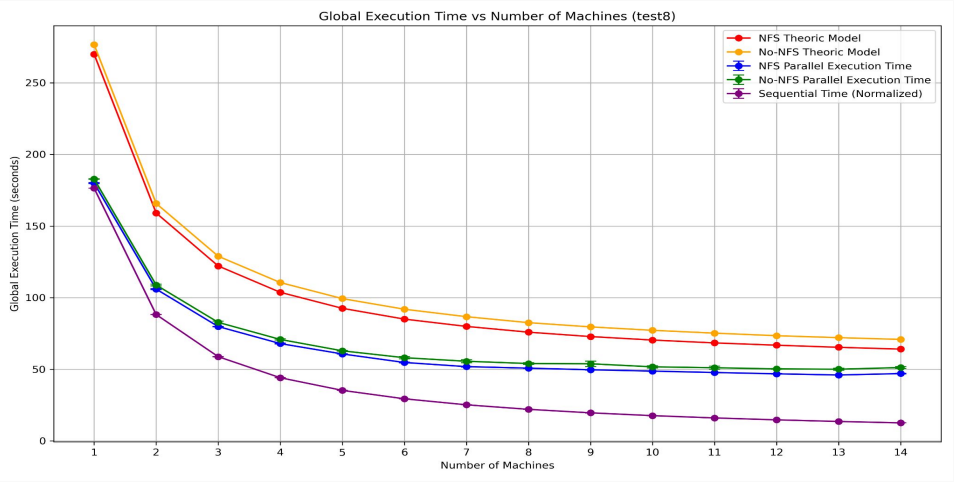
$$T_{\text{total}} = T_{\text{graph}}(N) + T_{\text{sparkConfig}} + \sum_{i=1}^{l} \left(\frac{\sum T[i] + \sum F[i]}{m} + \max(T[i] + F[i]) \right) + T_{\text{sparkOverhead}}$$

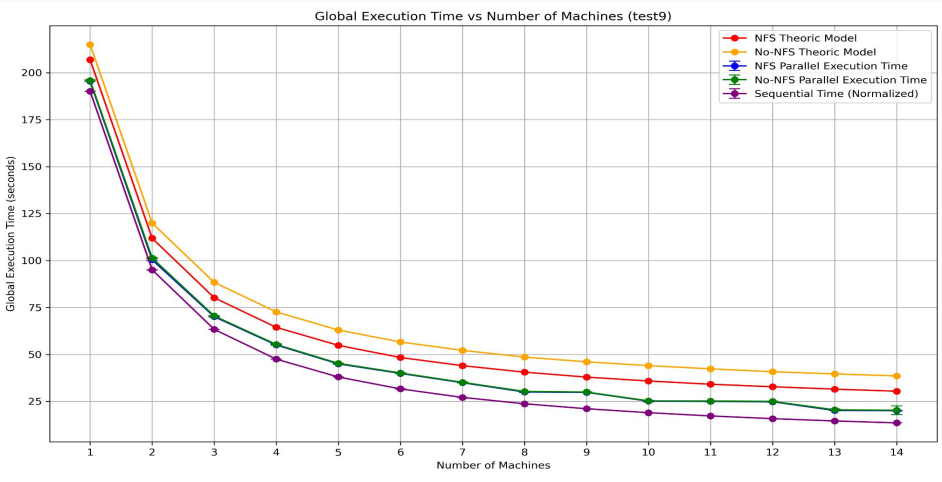
Instances et Performances

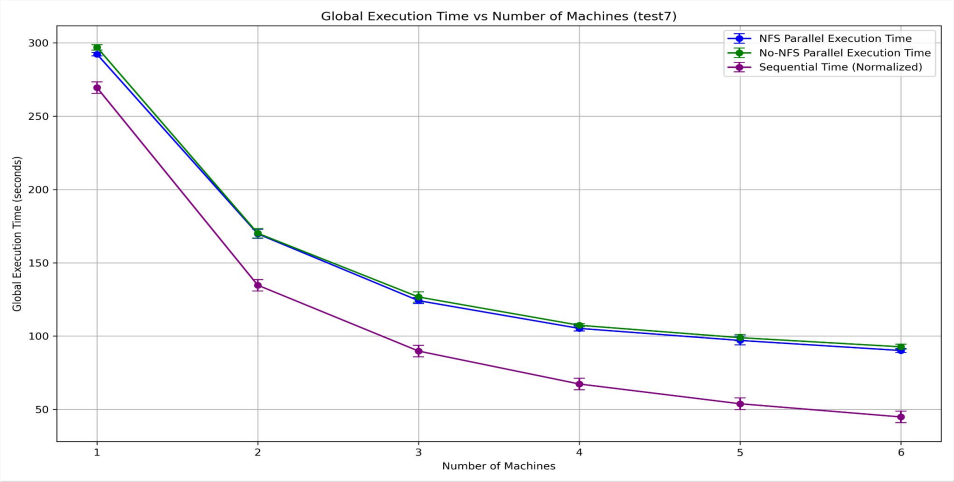
Tgraph et **Tsparkoverhead** sont prédits pour `un nombre de niveaux` et `de tâches par niveau` (**vecteur N**) donnés à l'aide de deux régressions linéaires entraînées sur des données expérimentales pour plusieurs versions d'un Makefile bien spécifiques. (Un script python génère le Makefile et collecte les mesures).

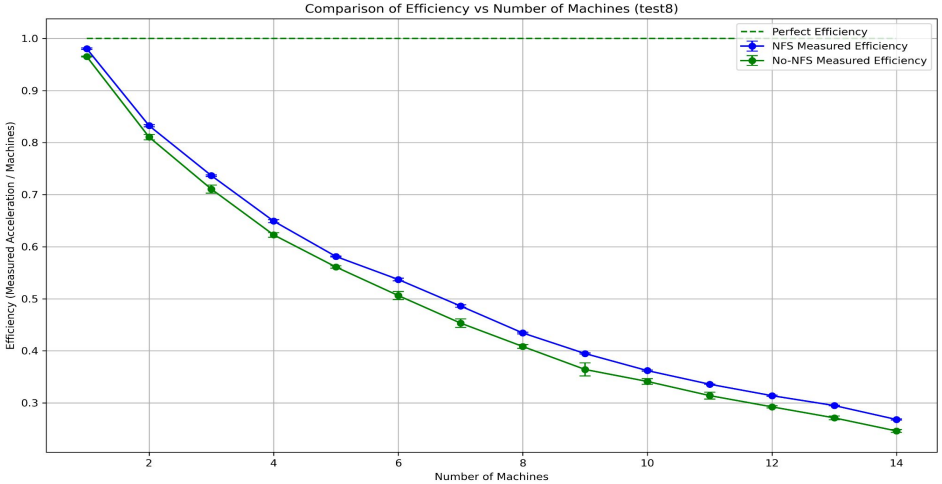
Le script génère pour **différents vecteurs N** un Makefile où chaque tâche d'un niveau i dépend de toutes les tâches des niveaux j < i, pour collecter les mesures au pire des cas.
Ensuite le script entraîne les deux modèles pour pouvoir ensuite prédire les valeurs de **Tgraph** et **Tsparkoverhead** dans le modèle théorique.

Test 7	***Test 8***	***Test 9***
Un Makefile plus réel pour comparer les performances des deux modèles entre eux avec des commandes réelles (*gcc* , *echo*) qui requiert des fichiers. 3 niveaux chacun de taille maximale 500 calculant les nombres premiers inclus dans un interval de taille aléatoire.	Un Makefile contenant un nombre de niveaux (4 pour notre instance) chacun contenant un nombre aléatoire de tâches (max = 2000 pour notre instance) avec une dépendance entière. Cette instance permet d'analyser le coup ajouter par le travail du driver (**Tsparkoverhead**) de la construction du graphe (**Tgraph**) pour le modèle Sans NFS.	Un Makefile pour comparer le modèle théorique aux résultats experimentaux Un seul niveau contenant 3952 (multiple de 104 = le nombre de coeurs par machine) tâche de durée exactement égale 5 secondes (sleep 5). Cela permet de prédire le fonctionnement parfait (le modèle théorique) du modèle et le comparer aux performances réelles.









Instances et Performances: tous les tests sont réalisés sur des machines paradoxe du site Rennes (104 coeurs)

+ Notre make Sans NFS est distribuée sur plusieurs sites.
+ Création d'un modèle hybride profitant de la localité des fichiers sur le même site.
+ Tolérances aux fautes au niveau des connexions entres les machines (implémentation d'un protocole simple d'accuse de reception) + re-exécution de la commande en cas d'échec.
+ Parallélisation de l'exécution des tâches et exploitation maximales des performances des machines.
+ Facilitation du déploiement et de l'exécution a l'aide de scripts robustes.
+ La distribution des tâches est optimale -> affecter une nouvelle tâche à une machine une fois qu'elle (un de ces coeurs) finisse sa tache.

- Exécution par niveau vu que spark ne supporte que les RDD.
- Exécution de toutes les tâches du niveau même si la première échoue.
- Le génération du Map<tâche, toutes les dépendances directe et indirectes> devient coûteux puisque c'est fait par le driver sur une seule machine, même en parallélisant l'algorithme (ce qui nous paraît optimal) sur 104 coeurs on ne peut pas gérer un Makefile à dépendance entière avec 5 niveaux chacun 3000 tâches + problème de OutOfMemory dans le tas du programme Java.
- A cause de l'isolation des coeurs sur chaque machine, dans certain cas de Makefile à dépendance entière va passer son temps à faire des copier coller (jusqu'à 400 000 copier coller + disc quota exceeded).
- Travail intensif sur le développement du modèle Sans NFS (Problèmes de synchronisation des coeurs, détection des fichier générés).

Recul sur notre travail