# Introduction :

Ce projet est divisé en deux parties :

La première est à propos de la conception d’une application Big Data qui sert à implanter et évaluer une méthode de descente de gradient pour la résolution d’un problème de filtrage collaboratif, en estimant les notes manquantes dans le jeu de données, afin de pouvoir proposer de futures recommandations de films pertinentes pour chacun des utilisateurs.

La deuxième partie sert à une étude de scalabilité de l’infrastructure du big data en comparant les différentes performances de l’exécution du programme, en changeant d’une part le nombre de slaves, et d’autre part la taille des blocks. Ce qui permet de déterminer la configuration optimale de l’infrastructure.

# Algorithme de descente de gradient

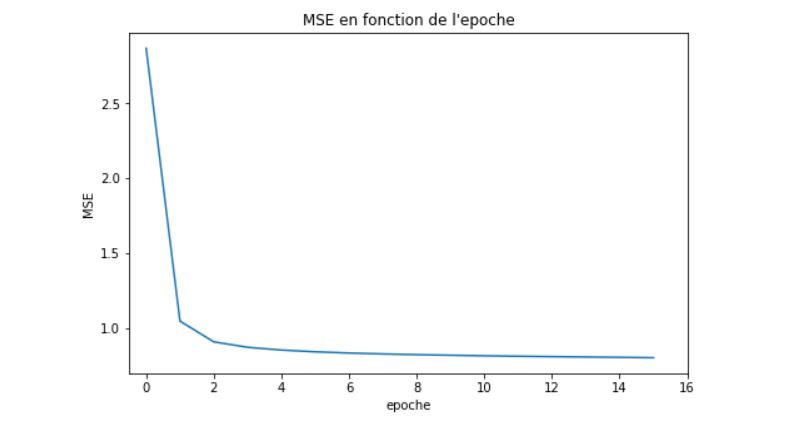


Figure . MSE en fonction de l'itération

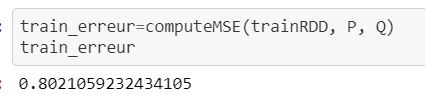
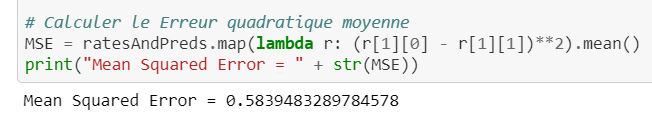


Figure . Résulta de MSE

Depuis la figure ci-dessus, on constate qu’il y a encore un écart entre notre modèle et le modèle de Pyspak.mllib, peut-être on doit augmenter le nombre des itérations pour obtenir des meilleurs résultats, pour l’instant on s’est limité à 15 itérations, puis on a vérifié que l’erreur de test et l’erreur d’entérinement sont presque le même, donc il n’y a pas de surajustement que on peut encore entrainer les données.

# III) Infrastructure/Spark avec Docker : Etude de scalabilité

Le but de cette partie est de faire une étude de scalabilité de l’infrastructure Spark avec Docker, en changeant, d’une part, le nombre de Slaves et d’autre part la taille des blocks et comparer le temps de l’exécution pour évaluer la performance de l’infrastructure.

## Analyse de l'impact du nombre des Slaves

On a essayé de changer le nombre de slaves en gardant la taille de blocks (16 Mb) fixe. Premièrement, on a constaté que le nombre des Slaves affecte la distribution de Block de données après avoir les copiées dans le HDFS système. Le nombre de Block total est 32 pour le fichier de 512 Mb pour les différents nombres de Slaves ; plus précisément pour un seul Slave, les 32 Blocks sont tous distribués dans le même nœud du Slave, mais quand on augmente le nombre de Slaves, les Blocks peuvent être distribués dans les autres nœuds disponibles, par exemple pour 4 Slaves les blocks sont distribués de la façon suivante : chacun des Slave 1 et Slave 3 contient 9 Blocks, ainsi que pour les Slave 2 et Slave 4 dont chacun contient 7 Blocks, et la somme au total est bien 32 blocks.

Les résultats obtenus sont regroupés dans la figure ci-après.

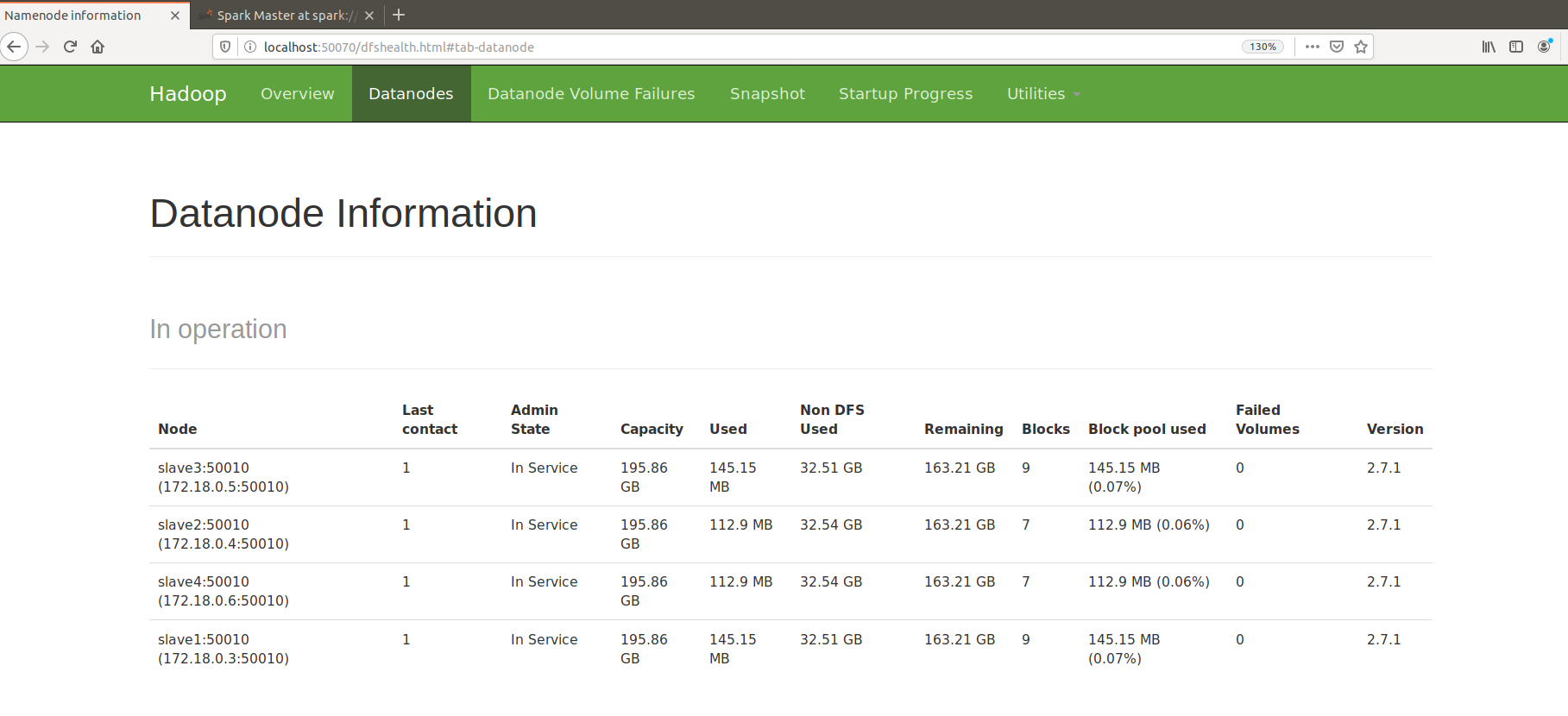
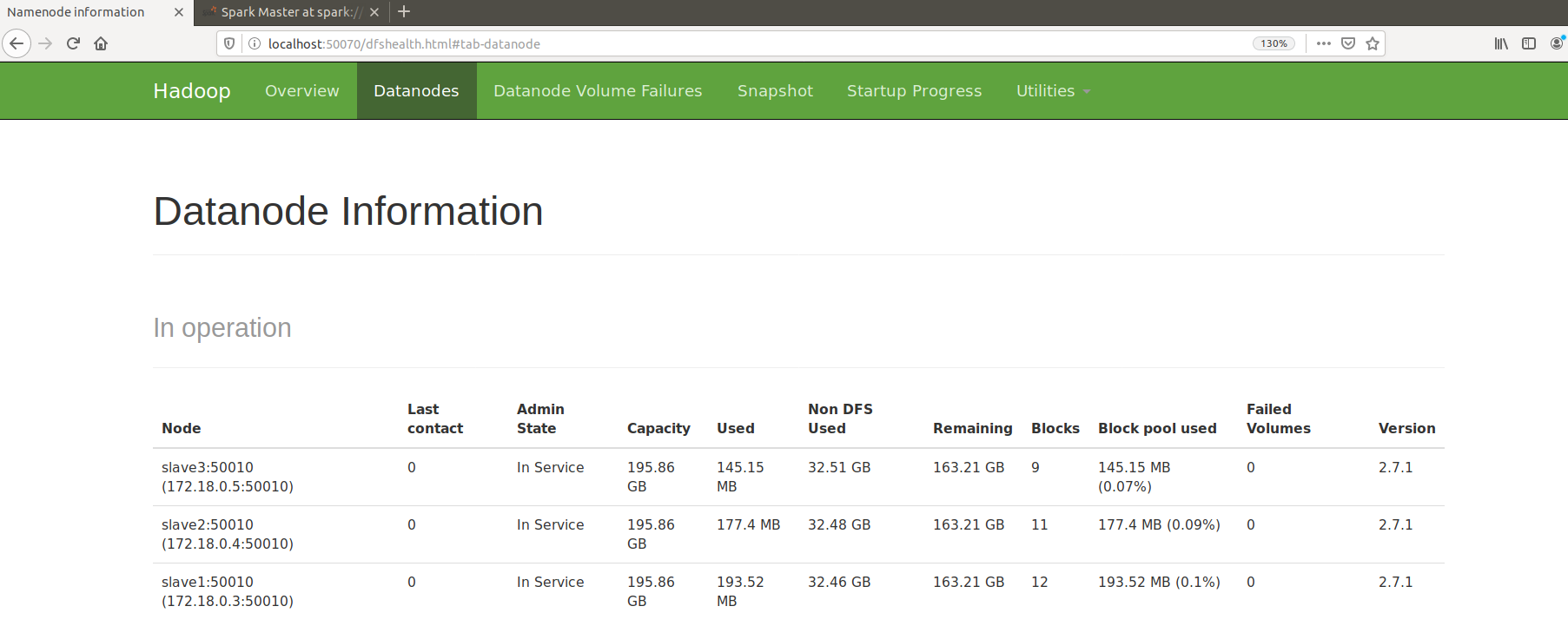
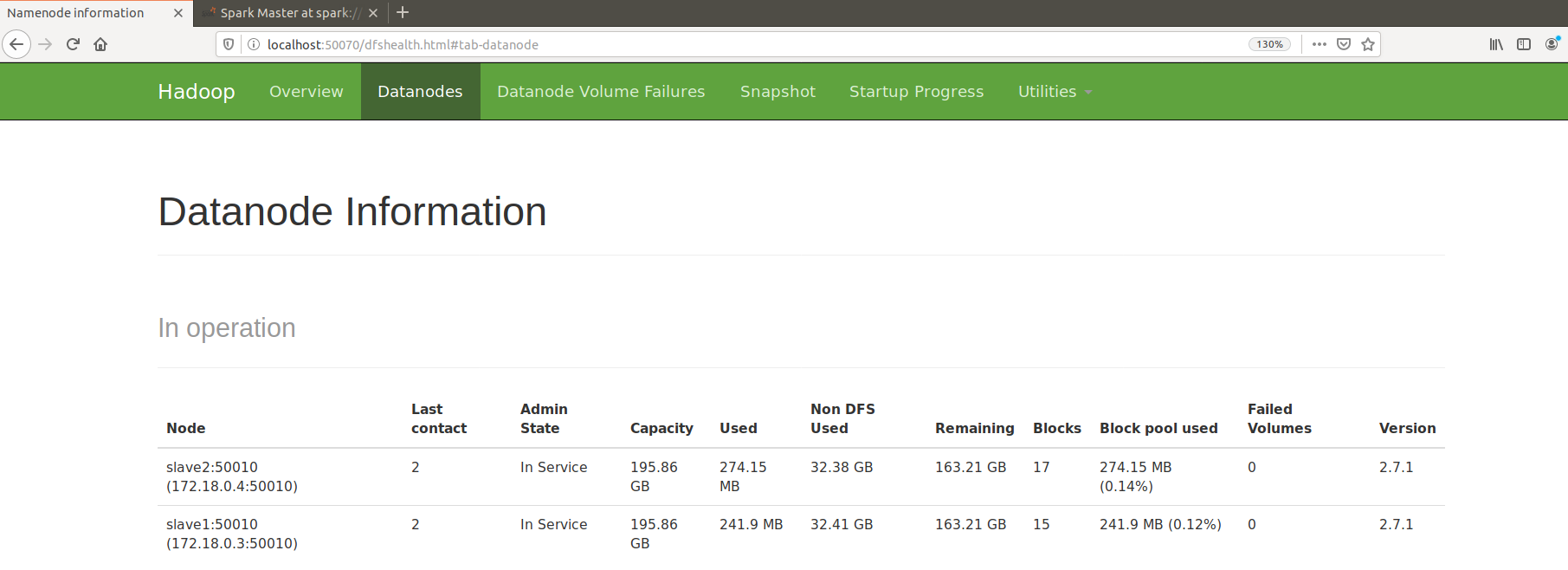
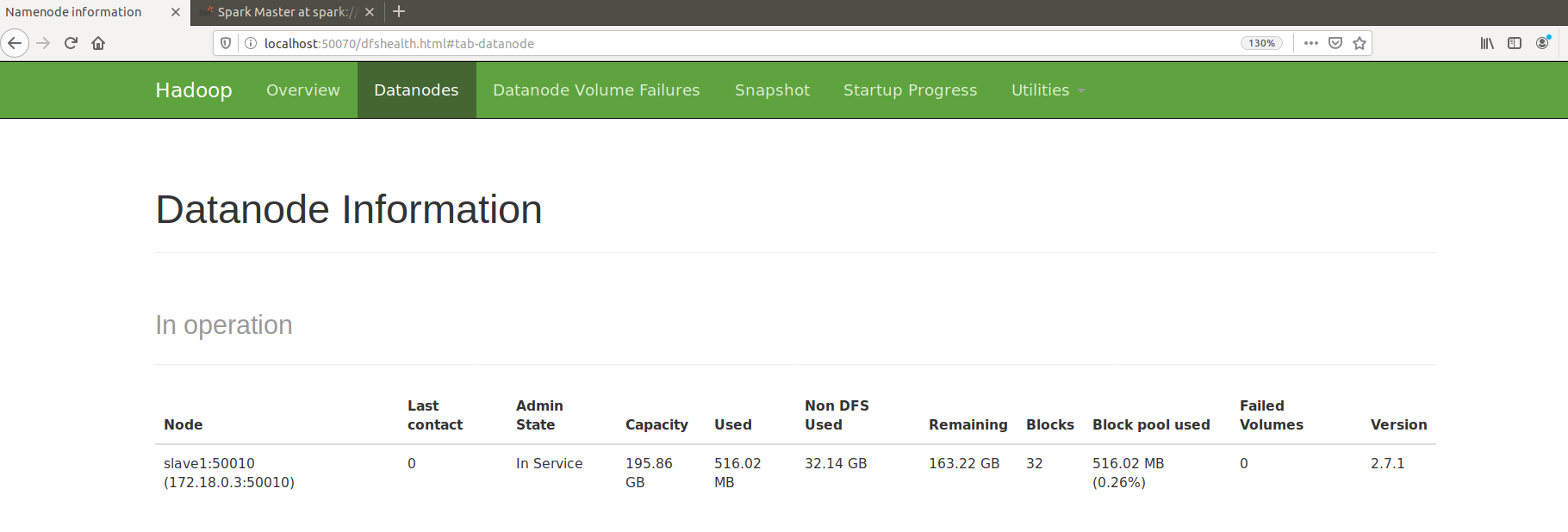


Figure 1. Informations de Datanode en fonction de différents nombres de Slaves

Après le lancement de programme de Word Count, en changeant le nombre de slaves on a obtenu différentes performances, qui sont regroupées dans les schémas ci-dessous.

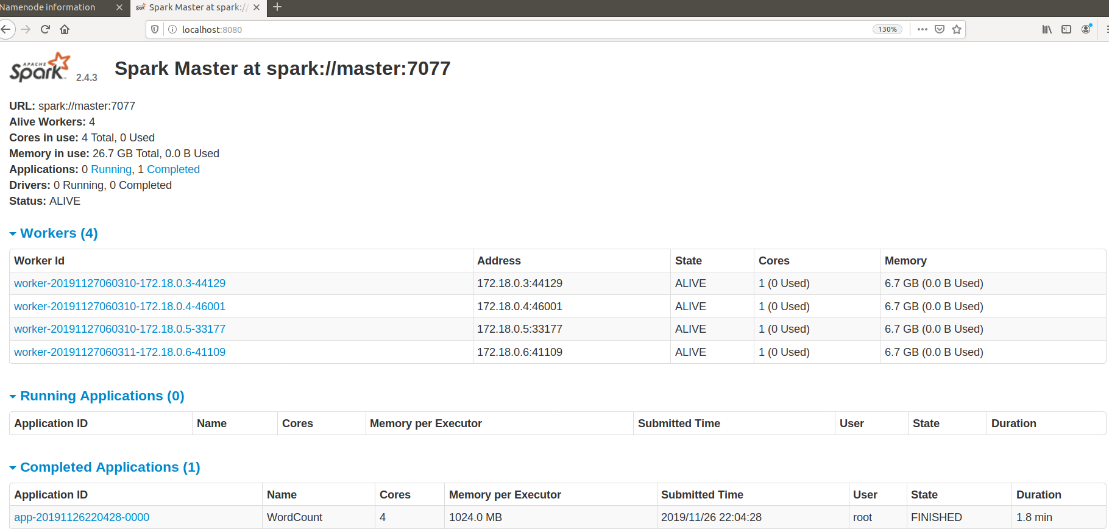
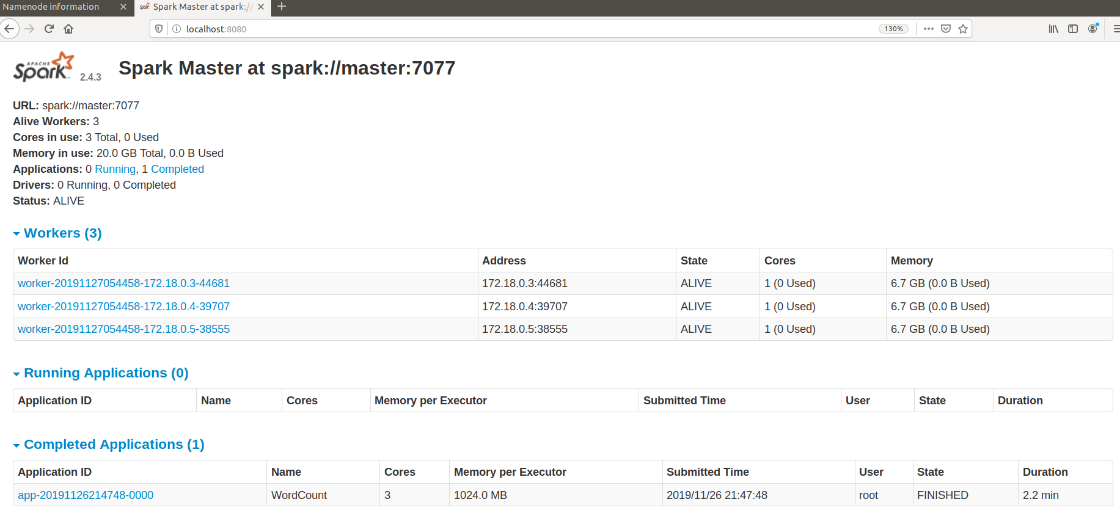
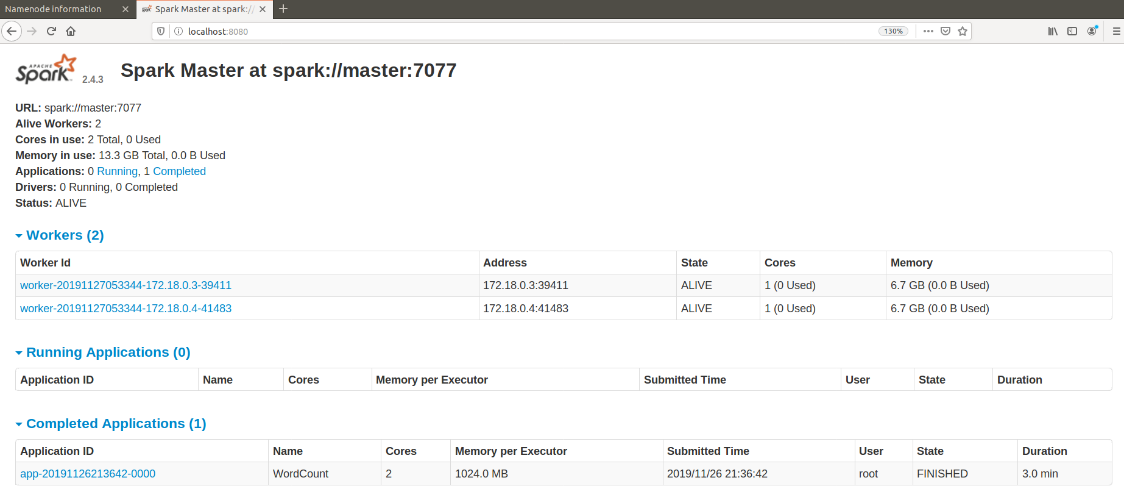
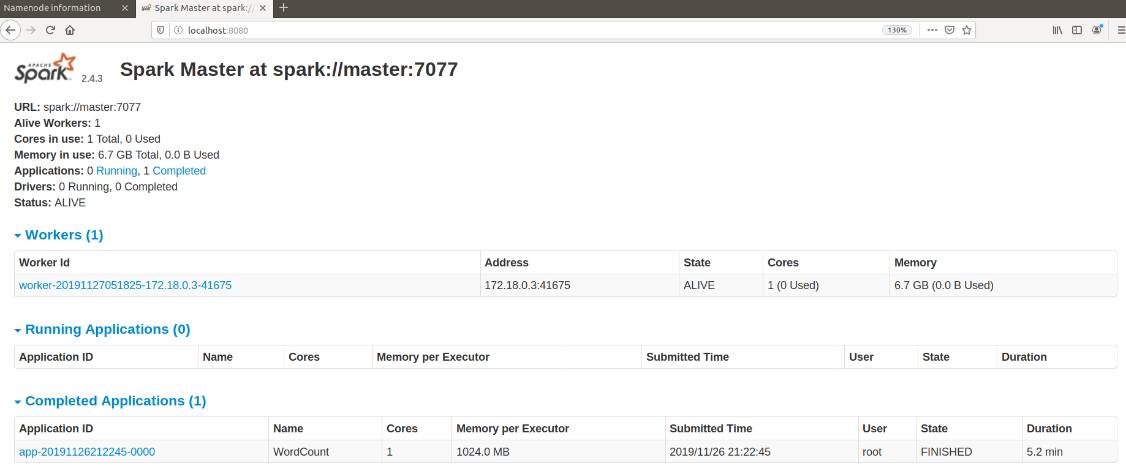


Figure 4. L'information de MasterNode en fonction du nombre différent de Slave

En comparant le temps de l’exécution pour les différents nombres de Slaves, on remarque qu’il diminue quand le nombre de slaves augmente, ce qui est logique, la performance du programme devient meilleure, parce que les slaves peuvent travailler en même temps, ce qui économise le temps de l’exécution.

## Analyse de l'impact de la taille de Block

On a fixé le nombre de Slaves en 4, pour évaluer la performance en changeant la taille des Block de 16 Mb à 1 Mb.

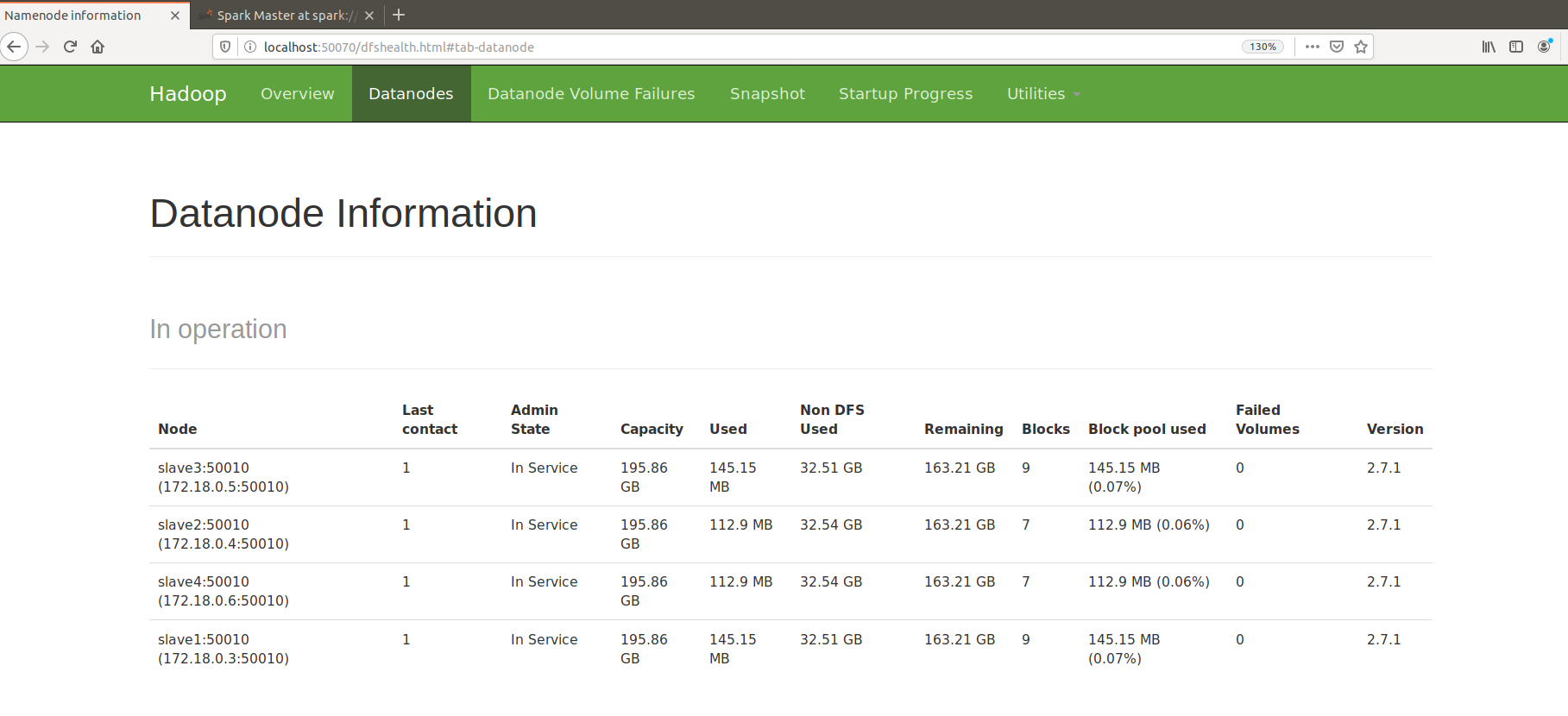


Figure 5. Block size 16 Mb Datanode

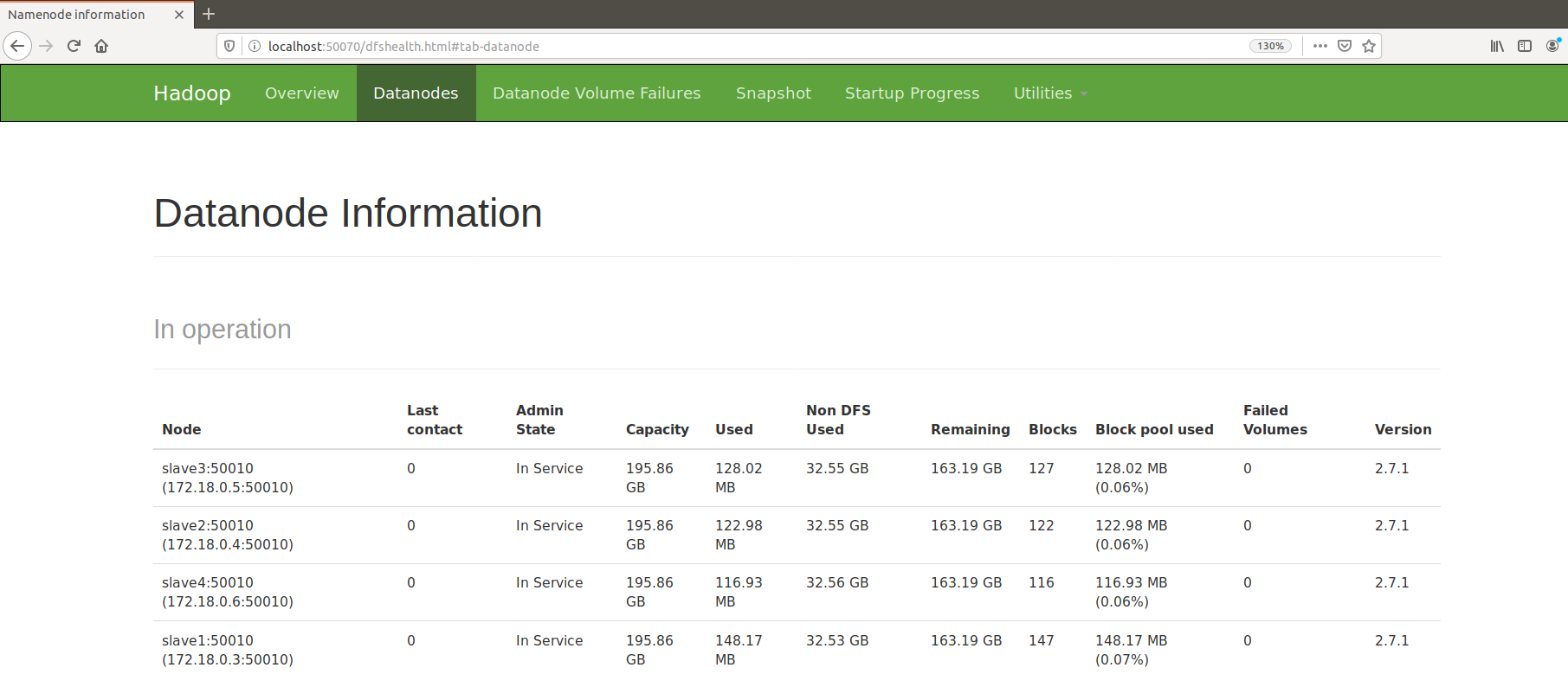


Figure 6. Block size 1 Mb Datanode

En comparant les deux cas, on a trouvé que le nombre des blocks augment beaucoup en diminuant leur taille, pour une taille de blocks égale à 1Mb, le nombre de blocks total est 512 qui est 16 fois plus grand au nombre correspondant à une taille de 16 MB, ce qui est cohérent et les données sont bien séparées en Blocks dans le système de HDFS, mais en perd en performance car plus on ajoute un block plus il faut relancer un map.

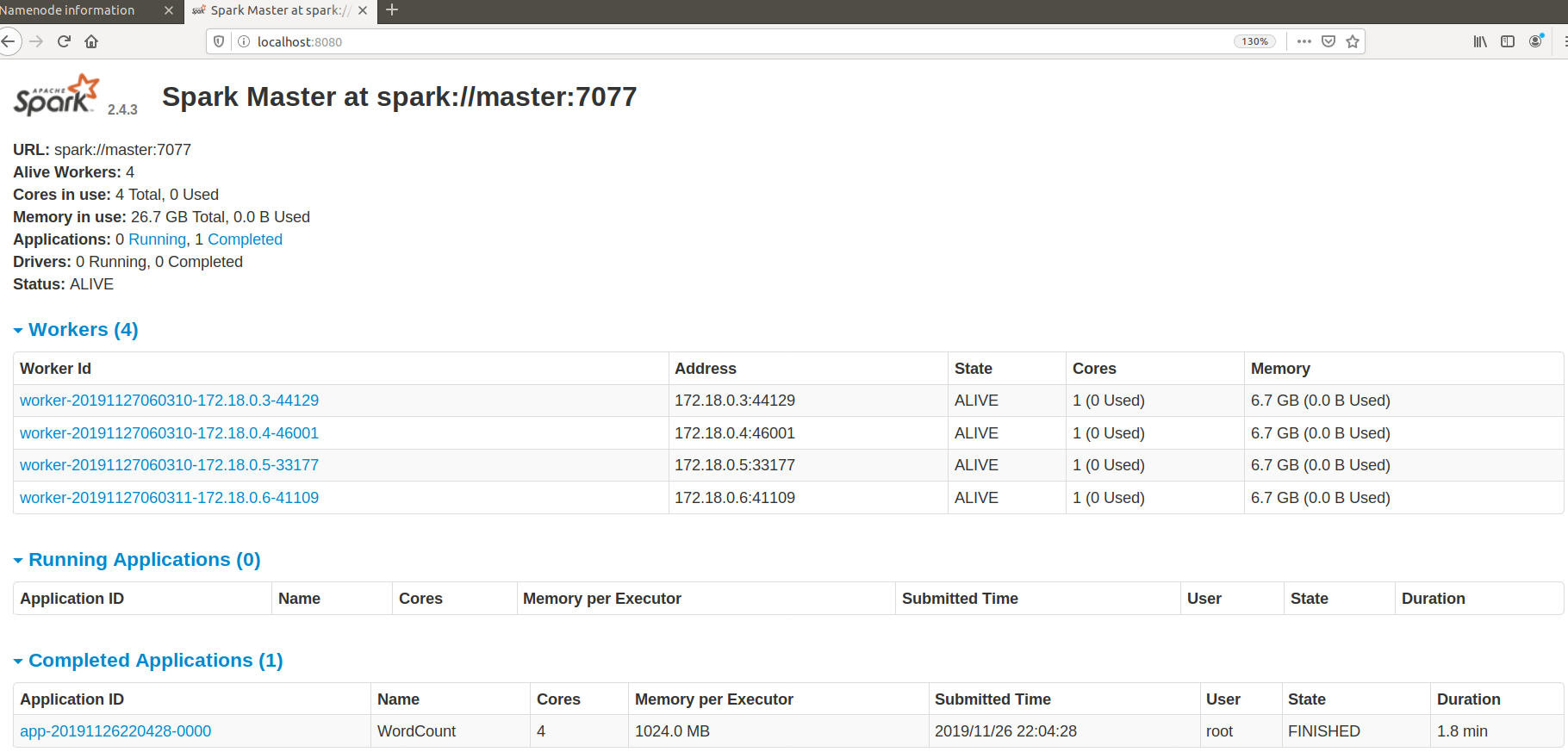


Figure 7. Block size 16 MB MasterNode

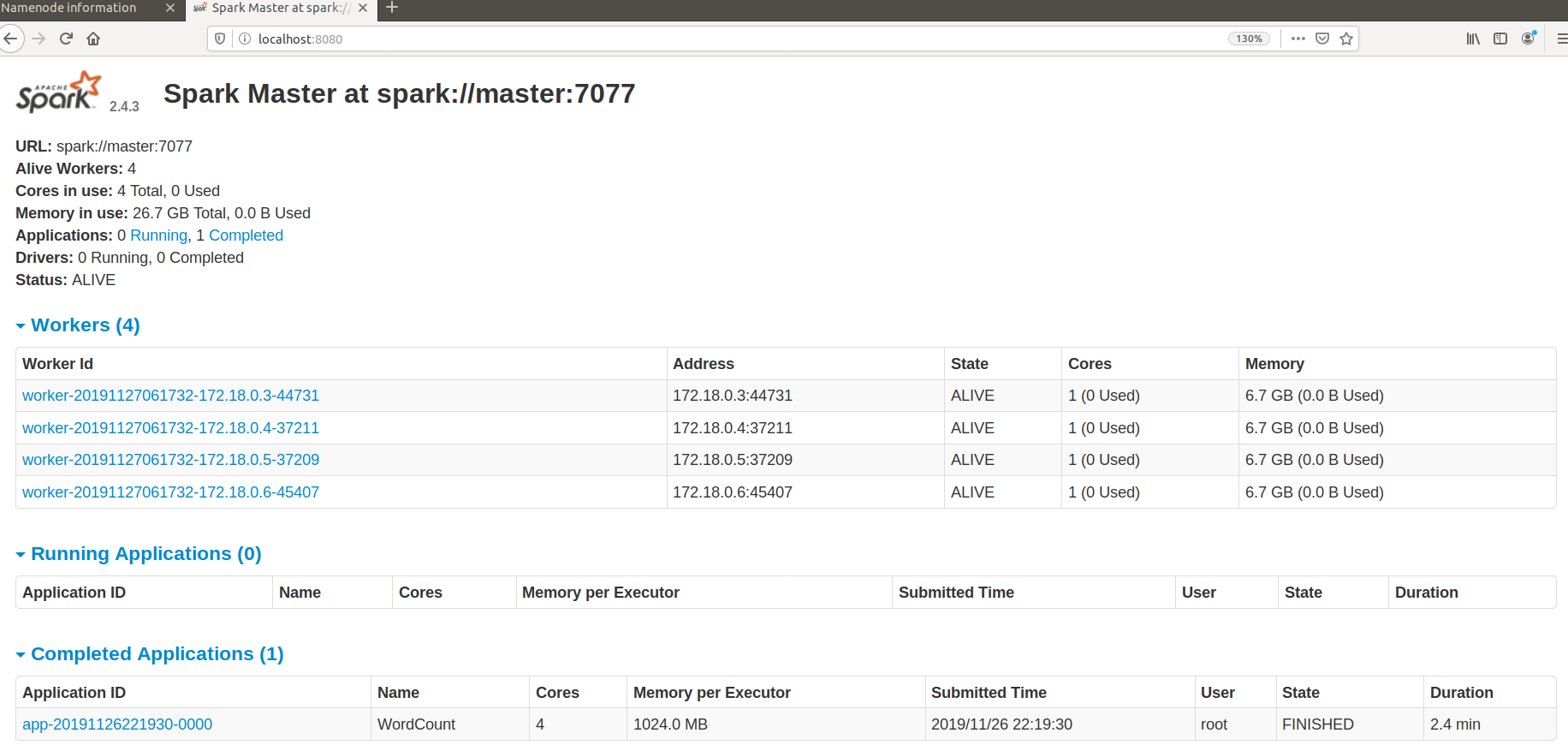


Figure 8. Block size 1 MB MasterNode

Avec une taille de block très petite, il a fallu plus du temps pour finir le fonctionnent. Parce que le temps d’exécution dépend de la taille de Block.

1. *Si La taille des blocks est trop petite*

La conception de HDFS a pour principe de prendre en charge les opérations de transmission des données de grande capacité en continu. Ainsi, même pour les opérations de lecture et d'écriture de données générales, la quantité de données impliquée est relativement importante, et si la taille est trop petite, il y aura plus de blocks de données à lire. Comme les blocks de données sont stockés de manière discontinue sur un disque dur, et pour les disques durs ordinaires l'adressage aléatoire est plus lent. Lorsque le temps de recherche mis par le disque dur est beaucoup plus long que le temps I/O(in/out), il devient alors un goulot d'étranglement dans le système. Une taille de block appropriée permet de réduire le temps de recherche du disque dur et d'améliorer le débit du système. De plus, pour HDFS, il n’a qu’un seul Namenoeude et sa mémoire est extrêmement limitée par rapport à Datanoeude, néanmoins le Namenoeude doit conserver l’information des Blocks dans le Datanode, s’il y a trop de Datanodes, il a besoin trop d’information pour la gestion des données stockées dans le Namenode, ce qui n’est pas bon pour la mémoire de Namenode.

1. *La taille des blocks est trop grande*

D’après l’infrastructure de MapReduce, les données doivent être rechargées au démarrage : plus le bloc de données est large, plus le temps de chargement des données est long et plus le processus de récupération du système est long. On a aussi des problèmes de complexité. La quantité de données est une relation linéaire de la complexité de la résolution des problèmes, pour un même algorithme, plus la quantité de données traitées est importante, plus sa complexité temporelle est grande. En plus des problèmes de temps réglementaire. Le Masternode surveille la situation des autres nœuds et chaque nœud rapporte périodiquement les mises à jour de travail et d'état terminées. Si un nœud reste silencieux pendant plus d'un intervalle de temps prédéfini, le nœud maître enregistre l'état du nœud comme mort et envoie les données allouées à ce nœud à d'autres nœuds. Donc, pour des blocs de tailles trop volumineux, cet "intervalle de temps prédéfini" n'est pas facile à estimer. Si la durée estimée est courte, le décès est mal évalué : dans le pire des cas, chaque nœud sera condamné à mort, et si le temps estimé est long, le temps d'attente sera trop long. Ce problème persiste car plus on a une taille de blocs grande moins on utilise les slaves.

# Conclusion

Ce projet nous a permis de mettre en œuvre l’importance de l’infrastructure dans le big data, et sa bonne configuration, les programmes qui sont tournées en séquentiel mettent plus de temps lors de leurs exécutions, alors que ceux tournés en parallèle prend moins de temps, et plus on ajoute des slaves plus on diminue le temps de l’exécution, mais par contre il faut des données de taille suffisants pour que cela soit toujours vrai, d’autre part parmi les paramètres les plus importants à bien configurer on trouve la taille de bloc, car plus on a des gros blocs moins on utilise les slaves, et si on fait des blocs plus petits on augmente leur nombre ce qui coûte aux entreprises, car à chaque fois on a un block on relance un map. Donc la bonne configuration permet aux entreprises de traiter des paquets de données qui arrivent tous les jours, en moins d’un jour d’où la nécessité de dimensionner l’infrastructure d’une taille qui permet de faire ces traitements, sinon on achètera plus de machine sans intérêt.