PAUX Jean-Baptiste | VERSTRAETEN Baptiste | DUBOIS Martin | MEHIDI Yanis | BORREMANS Gwendhal

Hidoop

1 Architecture

Depuis la nouvelle version, nous avons repris l'architecture de l'application. Nous avons en particulier changé le service Hidoop. Il se compose désormais des classes suivantes :

- DaemonMonitor qui implémente le RessourceManager. Son but est de superviser l'exécution des application sur l'ensemble de la grappe Hidoop. Il gère la liste des Jobs en cours d'exécution, récupère les erreurs des MapReduce et déclenche l'exécution des Jobs en RMI.
- DaemonDataNode est lancée exclusivement sur les machines dotées d'un service DataNode. Son but est d'exécuter les opérations de mapping sur les chunks d'un fichier situés sur la machine et d'enregister localement le résultat de l'opération de traitement pour chaque chunk. Il utilise une méthode de rappel lorsqu'il a terminé son traitement pour assurer un fonctionnement asynchrone, c'est la classe Job qui assure ce fonctionnement.
- Job est le coeur de chaque processus d'application lancé sur Hidoop. Il récupère la liste des DataNodes par une communication TCP avec le NameNode, il crée un méta-fichier sur le Hdfs qui correspond au fichier résultat de l'application et pour chaque DataNode, il lance un traitement de mapping. Il possède également une méthode de rappel lorsqu'un DaemonDataNode a fini son traitement et des méthodes de mesure pour l'évaluation de performance.
- HidoopServer est la classe qui combine le DaemonDataNode et le DaemonMonitor. C'est la classe qui doit être lancée pour mettre en place la grappe Hidoop.
- HidoopClient est à lancer pour exécuter une application sur Hidoop. Elle communique par RMI avec le HidoopServer exécutant le DaemonMonitor, en lui fournissant l'application et le nom du fichier existant sur Hdfs qu'elle souhaite exécuter. C'est également cette même classe qui s'occupe d'effectuer la réduction une fois l'ensemble des opérations de mapping terminées sur chaque DaemonDataNode, en allant récupèrer grâce au NameNode les fragements de chaque map sur les différents DataNode.

2 Points délicats

Le point le plus délicat du projet est la gestion des fichiers de grande taille sur le HDFS. En effet, la manière dont nous avons géré les fichiers lors de la première version utilisé la RAM de l'ordinateur. Ainsi dès que les tronçons de fichier dépassés 1Go en taille, l'application s'arrêtait pour manque d'allocation mémoire. Nous avons du modifier se comportement en envoyant petit à petit des tronçons de grande taille pour éliminer l'erreur. Un autre point qui semble délicat est la gestion des erreurs lorsque des pannes surviennent dans la grappe d'Hidoop. Nous avons couvert l'essentiel des

erreurs pouvant arriver mais nous n'avons traité que partiellement la mise en place de backup et la reprise sur erreur durant les opérations de MapReduce.

3 Application de MapReduce

Nous avons implanter une application permettant de calculer la valeur de pi avec l'algorithme de Monte Carlo.

Cette application est fortement CPU bound (car on ne fait qu'une lecture et 2 écritures pour le map et 2*Nombre_dataNodes lectures pour le reduce).

Nombre de points par DataNode	Pour 7 machines	Pour 4 machines
100 Millions	2,3 s (700Millions de points)	2,3 s (400 Millions de points)
500 Millions	10,7 s (3,5 Milliards de points)	10,58 s (2 Milliards de points)
700 Millions	18,15 s (4,9 Milliards de points)	14,85 s (2,8 Milliards de points)
1 Milliard	X	21,236 s (4 Milliards de points)

Pour un même traitement PAR datanode on obtient le même temps d'éxecution, donc plus on ajoute de DataNodes plus on pourras traiter de points totaux. Le speedup est ici directement proportionnel au nombre de DataNodes. Cela s'explique par le fait que le traitement du reducer est très simple, ajouter des dataNodes ne complique par les calculs et ajoute de la précision.

4 Analyse de performance

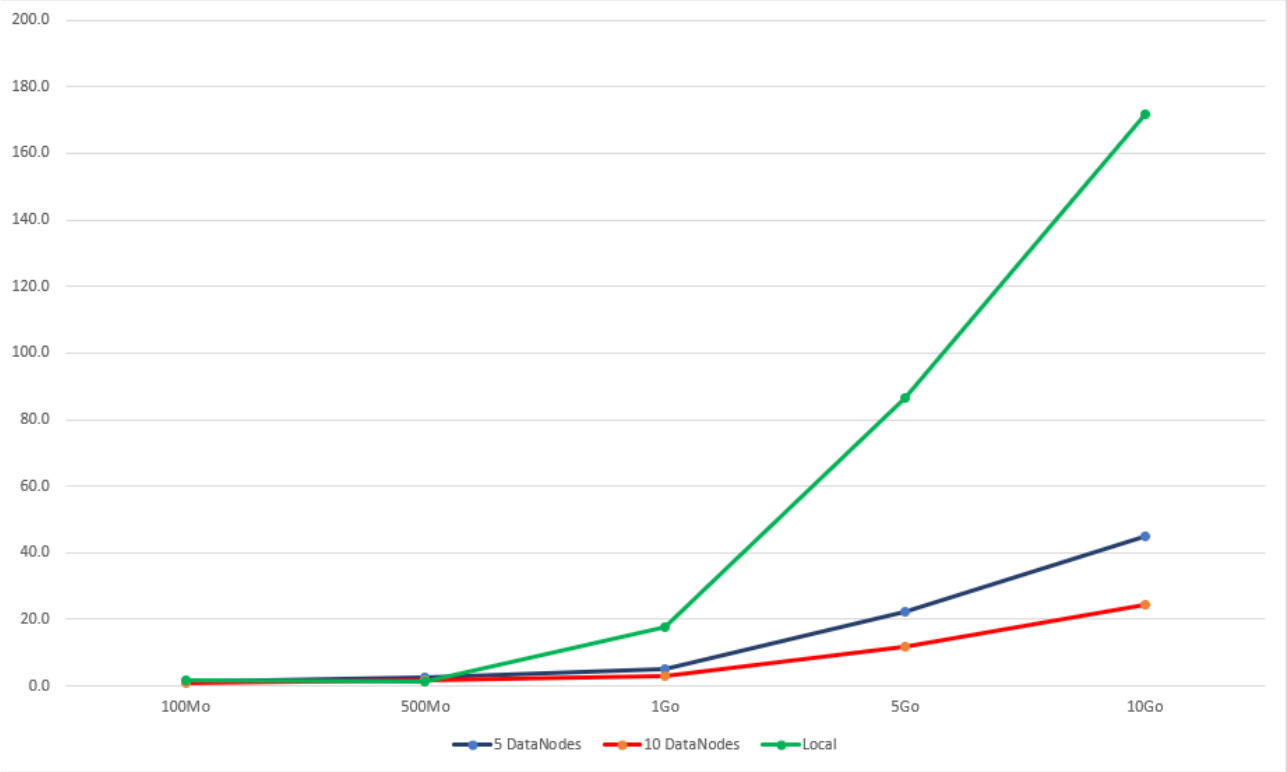
Pour notre analyse nous nous sommes limités à l'étude de l'application MyMapReduce du projet ainsi qu'à l'utilisation d'un unique réducteur pour l'opération de réduction.

Nous avons analysé les performances de notre Hidoop suivant 2 critères :

- Le nombre de machine dans la grappe Hidoop
- La taille du fichier sur lequel s'applique l'opération de MapReduce

Nous avons ainsi obtenu ce tableau ainsi que cette courbe :

Taille du fichier	Grappe de 5 machines	Grappe de 10 machines	Exécution séquentielle
100 Mo	1.088 s	1.078 s	1.729 s
500 Mo	2.663 s	1.765 s	1.153 s
1 Go	4.962 s	2.772 s	17.782 s
5 Go	22.182 s	11.589 s	86.68 s
10 Go	45.008 s	24.353 s	171.781 s



Nous pouvons remarquer dans un premier temps que l'exécution à l'aide d'une grappe de machine est beaucoup plus performante qu'une exécution séquentielle. De plus, l'augmentation du nombre de machines permet de réduire par 2 le temps d'exécution dans le cas où les fichiers sont très volumineux.

5 Conclusion

Grâce à notre Hidoop, simple en fonctionnalités, nous avons pu mettre en évidence l'intérêt d'avoir une répartition du calcul sur un ensemble de machines. La vitesse ainsi obtenue est largement supérieur à un calcul multi-coeur mais nécessite une architecture plus complexe. Il serait intéressant de comparer cette architecture avec l'architecture émergente des Many-Cores en terme de performance mais aussi en terme de consommation.