Annale

Quelle différence principale y a t'il entre un cluster HPC (High Performance Computing) pour le calcul parallèle et un cluster Big Data (devant exécuter des applications Hadoop) ? (1 point)

La différence principale entre un cluster HPC (High Performance Computing) et un cluster Big Data est :

1. **Objectif**:
   - HPC : Calcul intensif, simulations scientifiques.
   - Big Data : Analyse de grands volumes de données diverses.
2. **Architecture**:
   - HPC : Processeurs multi-cœurs puissants, beaucoup de RAM.
   - Big Data : Disques massivement parallèles, architecture distribuée.
3. **Performance**:
   - HPC : Mesuré en FLOPS (Floating Point Operations Per Second).
   - Big Data : I/O throughput et traitement distribué.

En résumé, l'HPC est pour le calcul rapide, tandis que le Big Data gère de grandes quantités de données.

Donnez deux raisons pour lesquelles les blocs sont répliqués dans HDFS ? (1 point)

Dans HDFS (Hadoop Distributed File System), les blocs sont répliqués principalement pour deux raisons :

1. **Fiabilité** : La réplication des blocs assure la tolérance aux pannes en cas de défaillance d'un nœud. Si un nœud tombe en
   panne, le système peut récupérer les données à partir des copies présentes sur d'autres nœuds.
2. **Performance**: Les répliques permettent une meilleure performance du système en offrant des options pour exécuter le
   traitement de données plus près des données stockées (data locality). Cela réduit la latence et augmente l'efficacité du traitement
   par Hadoop MapReduce ou autres frameworks.

A quoi correspondent des partitions dans l'exécution d'une application Spark (2 points)

Dans l'exécution d'une application Spark, les partitions jouent un rôle crucial en termes de gestion et de traitement des données :

1. **Distribution des Données** : Les partitions représentent la division du dataset distribué sur le cluster. Chaque partition est
   chargée dans la mémoire d'un nœud (executor) spécifique. Cette distribution permet à Spark de traiter les données en parallèle,
   améliorant ainsi l'efficacité et la rapidité des opérations.
2. **Optimisation du Traitement** : Les partitions déterminent comment les tâches sont exécutées sur le cluster. Une bonne
   stratégie de partitionnement peut réduire la quantité de transfert de données entre nœuds, optimiser l'utilisation des ressources
   et minimiser la latence dans le traitement. Spark utilise par défaut un nombre de partitions basé sur le nombre de cœurs
   disponibles, mais il est souvent nécessaire d'ajuster ce nombre pour une meilleure performance en fonction du dataset et des
   opérations effectuées.

Dans Spark, quelle différence essentielle y a t'il entre un map() et un reduceByKey() du point de vue des communications réseau ? (2 points)

1. **Map()**:
   - L'opération `map()` applique une fonction sur chaque élément du RDD (Resilient Distributed Dataset) indépendamment et
     localement. Elle ne nécessite aucune communication réseau entre les nœuds pour son exécution, car elle opère sur les partitions de
     données déjà présentes dans la mémoire des nœuds. Les résultats sont calculés de manière locale sans besoin d'échanger
     d'informations avec d'autres nœuds.
2. **ReduceByKey()**:
   - En revanche, `reduceByKey()` implique une phase de communication réseau significative. Cette opération est utilisée pour
     réduire les valeurs associées à des clés identiques dans un RDD. Lorsque `reduceByKey()` est appliqué, Spark effectue d'abord une
     étape de regroupement où toutes les paires clé-valeur avec la même clé sont rassemblées au sein du même partition. Cela nécessite
     un échange de données entre nœuds pour assurer que toutes les valeurs associées à une même clé se trouvent dans le même partition
     avant d'appliquer l'opération de réduction. Cette phase de shuffling est coûteuse en

termes de communication réseau, car elle
implique la redistribution des données à travers le cluster.

En résumé, `map()` ne nécessite pas de communication inter-nœuds et se déroule localement sur chaque partition, tandis que
`reduceByKey()` entraîne un shuffling et donc une communication entre les nœuds pour regrouper les données par clés avant
d'appliquer la réduction.

Expliquer la différence fondamentale entre Spark et Spark-streaming (2 points)

La différence fondamentale entre Spark et Spark Streaming repose sur leur objectif principal et leur manière de traiter les données
:

1. **Spark**:
   - Est un cadre général pour le traitement en mémoire des données à grande échelle.
   - Traite principalement des batchs de données, c'est-à-dire qu'il opère sur des jeux de données complets qui sont chargés dans
     la mémoire avant d'être traités. Cela permet une exécution rapide grâce au traitement en mémoire et à l'optimisation interne du
     framework.
2. **Spark Streaming**:
   - Est une extension de Spark conçue spécifiquement pour le traitement des flux de données en temps réel.
   - Divise les flux de données entrants en micro-batchs (tranches) qui sont traités séquentiellement à intervalles réguliers. Cela
     permet d'appliquer les mêmes transformations et actions disponibles dans Spark sur des données continues, offrant une solution pour
     le traitement événementiel et en temps réel.

En somme, tandis que Spark est adapté au traitement de grands volumes de données statiques ou batchs, Spark Streaming s'adresse à
la gestion et l'analyse de flux de données dynamiques en temps réel.

Résumez en une phrase ce qu'on entend par scalabilité dans Spark (2 points)

La scalabilité dans Spark se réfère à sa capacité à gérer efficacement des charges de travail croissantes en ajoutant simplement

plus de ressources au cluster, permettant ainsi un traitement proportionnellement rapide et stable des données.