Distributed Storage and Parallel Computing

Homework 2

12112627 李乐平

2.1 Please describe the difference between traditional MPI-based parallel computing and MapReduce-based parallel computing.

Solution:

In MPI, parallelization is achieved through explicit communication between nodes, where each node has its own memory space, and data is exchanged through messages. This requires careful synchronization and coordination among nodes. On the other hand, MapReduce simplifies parallelization by abstracting the programming model. It divides tasks into map and reduce phases, automatically handling distribution, fault tolerance, and load balancing. While MPI provides fine-grained control over communication, MapReduce offers a higher-level abstraction, making it more user-friendly but potentially less suitable for certain types of scientific or tightly-coupled computations.

2.2 Please describe the details of the pipeline of replicating data blocks.

Solution:

Firstly, the client creates file metadata in the Namenode. Subsequently, based on the topology distance provided by the Namenode, the client selects a nearby node in the Datanode to upload the file. After successful receipt of the file, the target node, in turn, selects a node on a different rack for backup. The node receiving the replication then chooses another node on the same rack as the initial target node for further backup. If the specified replica factor is greater than 3, subsequent replications are performed by randomly selecting nodes.

2.3 Is it permitted in MapReduce to modify the blocks with random writing? Why? What kind of writing is permitted?

Solution:

No. MapReduce is designed for WORM (Write Once Read Many) cases, only **append writing** is permitted. Random writes would introduce complexities in terms of synchronization and coordination among distributed nodes, making it challenging to maintain the simplicity and efficiency of the MapReduce paradigm.

2.4 Does MapReduce suit the machine learning paradigm? Why?

Solution:

Yes, the MapReduce framework can be applied to machine learning tasks, but its suitability depends on the nature of the machine learning algorithm. MapReduce was initially designed for large-scale data processing and parallel computation, and some machine learning algorithms can benefit from parallel computing.

Some machine learning tasks can be mapped to the two main stages of MapReduce: the Map stage and the Reduce stage. For example, in distributed training of deep neural networks, the training process for different batches of data can be distributed to multiple Map tasks, and their weight updates can be integrated through Reduce tasks. This can improve training speed, especially on

large-scale datasets.

However, not all machine learning algorithms are well-suited for the MapReduce framework. Some algorithms, especially those involving iterative optimization, may be impacted by the communication overhead and latency of MapReduce, as it is a discrete batch processing model. For these algorithms, frameworks more suitable for iterative computation, such as Apache Spark, might be preferable.