Li Xudong, Cao Kuangzheng, Qiu Yuyue

**Basic Tasks**

1. **File Replication on 20 Hosts**:
   * **Advantages**: Increased data availability, reduced access latency, and fault tolerance. With multiple replicas, if one host fails, data can still be accessed from others, ensuring high availability.
   * **Drawbacks**: Potential for data inconsistency if updates aren’t properly managed. Additionally, replication increases storage costs and management complexity.
2. **Processing a Large File on Remote Node**:
   * **Drawback**: Transferring large files over the network introduces delays and consumes bandwidth, impacting the overall performance.
   * **Better Approach**: Implement data locality by processing the data directly on the node where it’s stored. This can be achieved by using frameworks like Hadoop, which move computation to the data.
   * **Locality of Data**: This refers to minimizing the movement of data by processing it on the same node where it resides, thereby reducing network latency and enhancing performance.
3. **Sequential vs. Parallel Processing**:
   * **Sequential Processing**: Single-threaded execution of tasks one after another.
   * **Parallel/Distributed Processing**: Dividing a task into smaller sub-tasks that can be processed simultaneously across multiple cores or distributed nodes.
   * **Matrix Multiplication in Parallel**: For matrix multiplication, each element of the resulting matrix can be calculated independently by different cores or nodes. This can be achieved using multi-core processing or a distributed framework like Apache Spark.
4. **Parallel Execution of a Mathematical Formula**:
   * **Parallelizable Sections**: Identify independent operations such as (a + b), (c / d), (e \* f), and (g / h). These can be executed in parallel as they do not depend on each other.
   * **Limitations**: The combination steps, like subtracting results, must be executed sequentially, as they depend on the results of parallel sections.
   * **Node Execution Limitations**: If different nodes handle independent operations, data transfer between nodes becomes a bottleneck, leading to network delays.
5. **Finding the Largest Number in a Large File**:
   * **Parallel Approach**: Divide the large file into smaller chunks, each processed by a different node. Each node identifies the largest number in its chunk and sends the result to a central node that determines the global maximum.
6. **Difference in Parallel Approach**:
   * **Question 5 vs. Mathematical Formula**: In Question 4, the mathematical operations have dependencies between certain steps, while in Question 5, each node can independently find the maximum number without dependencies until the final reduction step.

**Medium Tasks**

1. **Hadoop Framework: HDFS**:
   * **Relationship Between NameNode and DataNode**: NameNode manages the metadata (directory structure, block locations) while DataNodes store the actual data blocks.
   * **Hadoop Cluster Example**:
     + **File Split Calculation**: A 180MB file divided into 64MB blocks results in three splits: S1 (64MB), S2 (64MB), and S3 (52MB).
     + **Number of Replicas**: Typically, HDFS maintains three replicas of each split for fault tolerance. Replication is achieved by distributing replicas across different nodes and racks.
     + **Configuration Example**: For split S1, store it on Node 1, Node 3, and Node 5.
     + **Handling Node Failures**: If Node 5 crashes, NameNode detects the failure and creates a new replica of S1 on another available node (e.g., Node 2) to maintain the replication factor.

**Advanced Tasks**

1. **MapReduce Framework**:
   * **JobTracker and TaskTracker Relationship**: JobTracker manages tasks and distributes them among TaskTrackers, which execute the tasks on individual nodes.
   * **Map Function**: Processes input data and generates key-value pairs.
   * **Reduce Function**: Aggregates or combines key-value pairs based on the key.
   * **Example for Counting Colored Squares**: The Map function emits a count of 1 for each color, and the Reduce function sums the counts for each color.
2. **MapReduce for Summing Quantities**:
   * **Problem**: You have data on customer orders, and you need to find the total quantity for each type of equipment.
   * **Input Key**: Order ID.
   * **Output Key**: Equipment Name (equip\_name).
   * **Map Function**: Emits (equip\_name, qty) pairs.
   * **Reduce Function**: Sums the quantities for each equip\_name.
3. **MapReduce for Streaming Service**:
   * **Problem**: Calculate the number of streams per film/TV program, paying for one stream per account.
   * **Map Function**: Emits (film\_id, account\_id) pairs, filtering out duplicates by account.
   * **Reduce Function**: Counts the number of unique account IDs for each film\_id, resulting in the number of paid streams.