Tema 4. Computación de altas prestaciones (HPC) y procesamiento a gran escala (Bigdata)

C1. Enumere tres características significativas del modelo de programación MPI

- Usado en Computación de altas prestaciones en sistemas de memoria distribuida.
- Sincronización de procesos. Compartición de datos mediante paso de mensajes.
- Ejecución en entornos heterogeneos

C2.- Marque cual de los siguientes modelos de programación son de memoria distribuida

Map-Reduce	SI	NO
MPI	SI	NO
OpenMP	SI	NO
Spark	SI	NO

C3.- Indique tres ventajas de utilizar Kubernetes

- Portabilidad y escalabilidad
- Ligero y mínimo uso de recursos
- Despliegue sencillo

C4.- ¿Qué es un RDD de Spark? ¿Qué características tiene?

Son las siglas de Resilient Distributed Data, y es la estructura básica para los datos en Apache Spark. Se trata de una colección de objetos disrribuidos que pueden ser transfomados en paralelo. El contenido de un RDD puede ser particionado y repartido en diferentes nodos de un cluster. Son Inmutables pero puedes crear un RDD aplicacando una trasformación a otro.

T1.- Al hablar de convergencia HPC y BigData, ¿cuál de las siguientes afirmaciones es más correcta?

a. Ambas tecnologías se ejecutan siempre sobre entornos virtualizados
b. Ambas tecnologías resuelven los mismos tipos de problemas
c. Ambas tecnologías usan los mismos modelos de programación
d. Ambas tecnologías se pueden complementar para el análisis intensivo y masivo de datos

T2.- ¿Qué es el top500?

a. La lista de las 500 máquinas más eficientes
b. La lista de las 500 máquinas más potentes
c. La lista de las 500 máquinas que menos consumen
d. La lista de las 500 máquinas que se usan en BigData y HPC

T3.- MPI es un paradigma de programación de sistemas…

a. con GPUs
b. de memoria distribuida
c. de memoria compartida
d. que utiliza Threads.

T4.- En BigData, ¿qué significa que un dato está en reposo?

a. Que no se mueve
b. Que está almacenado
c. Que se debe analizar en tiempo real
d. Que no cambia

T5.- A la hora de comparar Spark y Hadoop MapReduce, ¿cuál de las siguientes afirmaciones es incorrecta?

a. La ejecución de tareas en Spark son generalmente más rápidas que en Hadoop MapReduce
b. Ambos se puede programar en diferentes lenguajes
c. Hadoop MapReduce hace un uso más intensivo de memoria que Spark
d. Spark nace como mejora del modelo de programación MapReduce de Hadoop

How is Apache Spark different from MapReduce?

Apache Spark	MapReduce
Spark processes data in batches as well as in real-time	MapReduce processes data in batches only
Spark runs almost 100 times faster than Hadoop MapReduce	Hadoop MapReduce is slower when it comes to large scale data processing
Spark stores data in the RAM i.e. in-memory. So, it is easier to retrieve it	Hadoop MapReduce data is stored in HDFS and hence takes a long time to retrieve the data
Spark provides caching and in-memory data storage	Hadoop is highly disk-dependent

What is the significance of Resilient Distributed Datasets in Spark?

Resilient Distributed Datasets are the fundamental data structure of Apache Spark. It is embedded in Spark Core. RDDs are immutable, fault-tolerant, distributed collections of objects that can be operated on in parallel.RDD's are split into partitions and can be executed on different nodes of a cluster.

RDDs are created by either transformation of existing RDDs or by loading an external dataset from stable storage like HDFS.

What is a lazy evaluation in Spark?

When Spark operates on any dataset, it remembers the instructions. When a transformation such as a map() is called on an RDD, the operation is not performed instantly. Transformations in Spark are not evaluated until you perform an action, which aids in optimizing the overall data processing workflow, known as lazy evaluation.

What is shuffling?

Shuffling is the process of redistributing data across partitions that may lead to data movement across the executors. The shuffle operation is implemented differently in Spark compared to Hadoop.

Explain the types of operations supported by RDDs

Resilient Distributed Dataset (RDD) is a rudimentary data structure of Spark. RDDs are the immutable Distributed collections of objects of any type. It records the data from various nodes and prevents it from significant faults.

The Resilient Distributed Dataset (RDD) in Spark supports two types of operations. These are:

- RDD Transformation:
  The transformation function generates new RDD from the pre-existing RDDs in Spark. Whenever the transformation occurs, it generates a new RDD by taking an existing RDD as input and producing one or more RDD as output. Due to its Immutable nature, the input RDDs don't change and remain constant.
  Along with this, if we apply Spark transformation, it builds RDD lineage, including all parent RDDs of the final RDDs. We can also call this RDD lineage as RDD operator graph or RDD dependency graph. RDD Transformation is the logically executed plan, which means it is a Directed Acyclic Graph (DAG) of the continuous parent RDDs of RDD.

- RDD Action:
  The RDD Action works on an actual dataset by performing some specific actions. Whenever the action is triggered, the new RDD does not generate as happens in transformation. It depicts that Actions are Spark RDD operations that provide non-RDD values. The drivers and external storage systems store these non-RDD values of action. This brings all the RDDs into motion.
  If appropriately defined, the action is how the data is sent from the Executor to the driver. Executors play the role of agents and the responsibility of executing a task. In comparison, the driver works as a JVM process facilitating the coordination of workers and task execution.

What are the differences between regular FileSystem and HDFS?

- Regular FileSystem: In regular FileSystem, data is maintained in a single system. If the machine crashes, data recovery is challenging due to low fault tolerance. Seek time is more and hence it takes more time to process the data.
- HDFS: Data is distributed and maintained on multiple systems. If a DataNode crashes, data can still be recovered from other nodes in the cluster. Time taken to read data is comparatively more, as there is local data read to the disc and coordination of data from multiple systems.

Why is MapReduce slower in processing data in comparison to other processing frameworks?

MapReduce is slower because:

- It is batch-oriented when it comes to processing data. Here, no matter what, you would have to provide the mapper and reducer functions to work on data.
- During processing, whenever the mapper function delivers an output, it will be written to HDFS and the underlying disks. This data will be shuffled and sorted, and then be picked up for the reducing phase. The entire process of writing data to HDFS and retrieving it from HDFS makes MapReduce a lengthier process.
- In addition to the above reasons, MapReduce also uses Java language, which is difficult to program as it has multiple lines of code.

How does MPI compare to other parallel computing models, like OpenMP?

MPI and OpenMP are both parallel computing models, but they differ in their approach. MPI is a message-passing model designed for distributed memory systems. It allows processes to communicate by sending and receiving messages, making it suitable for large-scale computations on supercomputers.

On the other hand, OpenMP is a shared-memory model that uses threads for parallelism within a single node. It's simpler to use than MPI as it doesn't require explicit communication between threads. However, its scalability is limited compared to MPI due to its reliance on shared memory.

While MPI can be used alone or combined with OpenMP for hybrid parallelism, choosing between them depends on the problem size, hardware architecture, and programmer expertise.

Can you explain the difference between point-to-point communication and collective communication in MPI?

Point-to-point communication in MPI involves data transfer between two processes, typically using send and receive operations. It's useful for custom or irregular data patterns but can be complex to manage due to the need for matching sends and receives.

Collective communication, on the other hand, involves all processes within a communicator. Operations include broadcast, gather, scatter, reduce, etc., which are simpler as they don't require explicit pairing of send/receive calls. However, they're less flexible than point-to-point communications because they impose a specific pattern of interaction among processes.

How would you tackle the issue of load balancing in an MPI program?

Load balancing in MPI can be addressed by dynamic task assignment. Instead of statically assigning tasks to processes at the start, tasks are dynamically assigned as processes become available. This approach ensures that all processors are kept busy and no processor is idle while others are still working.

To implement this, a master-worker model can be used where one process (the master) distributes tasks to other processes (workers). The master sends a task to a worker, waits for it to finish, then sends another task. If a worker finishes its task quickly, it will receive a new task sooner, ensuring efficient use of resources.

Another method is work stealing, where an idle process requests tasks from a busy process. This requires careful synchronization to avoid conflicts but can result in better load balance if implemented correctly.

How is a non-blocking communication beneficial in MPI programming?

Non-blocking communication in MPI programming is beneficial as it allows for overlap of computation and communication, enhancing efficiency. It enables a process to initiate data transfer and then proceed with other tasks without waiting for the completion of the transmission. This asynchronous nature reduces idle time, especially in large-scale parallel computing where data dependencies can cause significant delays if processes are blocked until all communications complete. Non-blocking operations also provide better load balancing by allowing slower processes to catch up while faster ones continue working.

Can you illustrate how you would use MPI_Reduce in a problem-solving scenario?

MPI_Reduce is a collective operation that combines values from all processes and distributes the result back to all processes. It's useful in scenarios where we need to perform operations on distributed data.

Consider a scenario where multiple processes are calculating pi using the numerical integration method (as shown in laboratory). Each process calculates its area, and then, using MPI_Reduce, these "local" areas are reduced to "global" area to compute pi value.

TRUE/FALSE questions:

MPI programs may be SPMD …… TRUE/FALSE
Reduction operations are not parallelizable ……. TRUE/FALSE
Data locality is a mayor performance challenge …….. TRUE/FALSE
Caching is used to help overcome the memory bottleneck TRUE/FALSE