Review Questions: Parallelism in Data Engineering for ML

DSC 208R - Data Management for Analytics

June 2025

Questions and Answers

From "Introduction to Parallelism Review Questions"

Question 1

Briefly explain two advantages of large-scale data for analytics.

Answer: Large-scale data offers several advantages for analytics:

1. It allows for the study of granular phenomena in sciences and businesses that were previously impossible to observe or analyze.
2. It enables new applications and facilitates personalization or customization, such as tailored recommendations in e-commerce.
3. It supports more complex Machine Learning (ML) prediction targets and helps mitigate variance to achieve high accuracy.

Question 2

What are the "3 Vs" of so-called "Big Data"?

Answer: The "3 Vs" of "Big Data" are:

- **Volume**: Data that is larger than what can be processed or stored on a single node's DRAM.
- **Variety**: Data that comes in diverse forms, including relational data, documents, social media posts, and multimedia.
- **Velocity**: Data generated at a high rate, such as from sensors or surveillance systems.

Question 3

Why is "Big Data" a thing nowadays even though Petabyte-scale databases have been around since the 1990s?

Answer: "Big Data" has gained prominence now primarily due to two factors beyond just volume:

1. **Applications**: A new "data-driven mentality" has emerged across nearly all human endeavors, from web services like e-commerce and social media to scientific research, medicine, logistics, finance, and humanities. This widespread adoption drives the need to process and analyze massive, varied datasets.
2. **Storage and Compute Advancements**: While petabyte-scale databases existed, modern hardware advancements in storage capacity (HDDs, SSDs, NVM-NAND), compute capacity (multi-core CPUs, GPUs, TPUs), and DRAM capacity have made it feasible and cost-effective to process and manage these large datasets. Cloud computing has also democratized access to this powerful infrastructure.

Question 4

What is a dataflow graph? Name two data processing tools that deal with dataflow graphs:

Answer: A **dataflow graph** (also known as a Logical Query Plan in database systems or a Neural Computational Graph in ML systems) is a Directed Acyclic Graph (DAG) where vertices represent operators (e.g., relational algebra operators or tensor algebra operations), and edges represent the flow of data between these operators.

Two data processing tools that deal with dataflow graphs are:

1. **Relational Database Management Systems (RDBMSs)**: They internally convert SQL queries into Logical Query Plans, which are a form of dataflow graphs.
2. **Apache Spark**: While not explicitly named in the context of "dataflow graph" definition, Spark's RDDs and DataFrames often build execution plans that are essentially dataflow graphs. (Additionally, tools like **Dask** also manage computations as dataflow/task graphs).

Question 5

What is the basic difference between task-parallel systems and data-parallel systems?

Answer: The basic difference lies in how the workload is distributed and processed:

- **Task-parallel systems** (or functional parallelism) focus on dividing a larger problem into multiple independent tasks, where each task performs a different operation. These tasks can then be executed concurrently, often on different data or different parts of a larger computation. An example is a pipeline of different operations.
- **Data-parallel systems** (or data-level parallelism) involve applying the same operation or program to different partitions of a large dataset simultaneously. The data is sharded across multiple workers, and each worker performs the same computation on its local data subset.

  Essentially, task parallelism means "different things to different data," while data parallelism means "same thing to different data."

From "Data Parallelism Review Questions"

Question 1

How is data-parallelism different from task parallelism?

Answer: Data parallelism and task parallelism represent two distinct approaches to parallelizing computations:

- **Data Parallelism**: This involves partitioning a large dataset across multiple processors or nodes, with each processor executing the *same operation* or program on its own subset of the data simultaneously. The focus is on distributing the data. Examples include MapReduce and parallel RDBMS queries.
- **Task Parallelism**: This involves decomposing a problem into multiple distinct, independent tasks or functions. Each task can perform a *different operation*, and these tasks are then executed concurrently on different processors. The focus is on distributing the work (tasks). An example would be a processing pipeline where different stages run in parallel.

Question 2

What are the three main paradigms of multi-node parallelism in cluster computing?

Answer: The three main paradigms of multi-node parallelism in cluster computing are:

1. **Shared-Nothing Parallelism**: Each CPU (worker) has its own dedicated memory and disk, with communication occurring explicitly over a network. Data is typically partitioned (sharded) across these independent nodes.
2. **Shared-Disk Parallelism**: Multiple CPUs share access to the same set of disks via an interconnect, but each CPU has its own private memory.

3. **Shared-Memory Parallelism**: Multiple CPUs share access to a common memory pool and often a common set of disks, usually within a single machine due to memory coherence and bandwidth limitations.

Question 3

What are the three main data partitioning strategies in data-parallelism? How do they relate to data layout strategies?

Answer: The three main data partitioning strategies for row-wise (horizontal) partitioning in data-parallelism across k nodes are:

1. **Round-robin**: Tuple i is assigned to node $i \pmod k$.
2. **Hashing-based**: Requires one or more attributes to be designated as hash partitioning attributes. This is very common in practice for RA and SQL workloads.
3. **Range-based**: Requires one or more ordinal partitioning attributes and is often good for range predicates in RA/SQL.

These strategies relate to data layout by physically distributing data blocks or tuples across different nodes or disks in a cluster. Beyond row-wise partitioning, data can also be laid out using **columnar partitioning** (partitioning data by columns across nodes) or **hybrid/tiled partitioning** (a combination of row-wise and columnar approaches). These strategies determine how the data is physically stored and accessed in a distributed environment to optimize for different types of queries or workloads.

Question 4

What is the main difference of peer-to-peer cluster architecture vs. manager-worker architecture? Name three example systems of each.

Answer: The main difference between peer-to-peer and manager-worker cluster architectures lies in their centralization of control:

- **Manager-Worker Architecture**: This is a centralized architecture where one or a few special nodes (Managers/Masters) dictate what the other worker nodes should do and when they should communicate. The manager coordinates tasks, distributes work, and collects results. **Examples**: Dask, Apache Spark, and parallel RDBMSs.
- **Peer-to-Peer Architecture**: This is a decentralized architecture where there is no single manager node. Workers communicate directly with each other to coordinate tasks and exchange data. **Example**: Horovod. (Note: The provided materials explicitly name only one example for peer-to-peer. Other systems like BitTorrent, or certain blockchain networks, also exhibit peer-to-peer characteristics, but are not mentioned in the provided context.)

Question 5

Briefly explain 2 reasons why BSP may not yield linear speedup.

Answer: Bulk Synchronous Parallelism (BSP) often does not yield linear speedup due to several overhead factors:

1. **Manager Overhead**: The manager node incurs overhead in dividing up work among workers and collecting or unifying partial results from them. This centralized coordination can become a bottleneck as the number of workers increases.
2. **Stragglers**: BSP involves barrier synchronization, meaning the manager must wait until all workers have completed their current task before proceeding to the next piece of work. If even one worker is significantly slower (a "straggler"), it can delay the entire computation, limiting the overall speedup.
3. **Communication Costs**: The time and resources spent on communication between the manager and workers, and potentially among workers themselves, add to the total execution time and reduce the efficiency of parallelization.
4. **Per-Worker Overhead**: Each worker incurs startup and tear-down costs, which can become significant in fine-grained parallel tasks.

Question 6

Briefly explain 1 pro and 1 con of NFS vs. HDFS.

Answer: Based on the provided materials, information on NFS is not explicitly detailed for a direct comparison. However, the characteristics of HDFS are discussed. **Hadoop Distributed File System (HDFS):**

- **Pro**: HDFS is highly scalable, designed to scale to thousands of nodes and petabytes of data, and is fault-tolerant through data replication.
- **Con**: HDFS primarily supports read-only access and batch-append operations. It does not offer efficient fine-grained updates or writes to existing data blocks, making it less suitable for workloads requiring frequent modifications to existing files.

(For a full comparison with NFS, additional information on NFS's characteristics, such as its POSIX compliance and fine-grained write capabilities, would be needed, which is beyond the scope of the provided documents.)

Question 7

Does data-parallelism apply to dataflow graphs or task graphs?

Answer: Data parallelism primarily applies to **dataflow graphs**. In dataflow graphs, the operations are applied to streams or sets of data, which aligns perfectly with the concept of applying the same operation to different partitions of data. The MapReduce model, for instance, is a data-parallel dataflow framework. While data-parallel computations can be represented within broader task graphs (which can be more coarse-grained and show dependencies between larger computational tasks), the core concept of data parallelism itself maps directly to the data transformation nature of dataflow graphs.