Data Parallelism Part 2: Comprehensive Review

DSC 208R - Data Management for Analytics

Bulk Synchronous Parallelism (BSP)

Bulk Synchronous Parallelism (BSP) is the most common protocol for data parallelism in modern data systems, including parallel RDBMSs, Hadoop, and Spark. It typically combines shared-nothing sharding with a manager-worker cluster architecture.

The steps in a BSP model are as follows:

1. A sharded data file resides on workers.
2. A client submits a program (e.g., an SQL query, ML training) to the manager.
3. The manager divides the first piece of work among the workers.
4. Workers operate independently on their respective data partitions. Cross-talk between workers can occur if directed by the manager.
5. Each worker sends its partial results to the manager after completing its assigned task for that piece of work.
6. The manager waits until all k workers have completed and sent their results (barrier synchronization).
7. For the next piece of work, the process returns to step 3.

Speedup Analysis / Limits of BSP

Speedup Definition

Speedup in parallel processing is defined as:

$$\text{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given k(¿1) workers}}$$

Ideally, with k workers, the speedup would be k. However, in practice, this ideal is rarely achieved due to various overhead factors.

Cluster Overhead Factors Affecting Speedup

Several factors contribute to reduced speedup in a BSP cluster:

- **Per-worker overhead**: Includes startup and tear-down costs for each worker.
- **Manager overhead**: Involves the time taken by the manager to divide work among workers and to collect/unify partial results from them.
- **Communication costs**: The time and resources spent on communication between the manager and workers, and occasionally among workers themselves (when instructed by the manager).
- **Stragglers**: Barrier synchronization, a key part of BSP, can suffer from "stragglers" – slow workers that delay the progress of the entire computation, as the manager must wait for all workers to complete before proceeding to the next stage.

Hadoop Distributed File System (HDFS)

HDFS is a prominent example of a distributed file system designed for storing very large files across clusters of commodity hardware. It is built on a shared-nothing architecture for data, employing replication for fault tolerance.

Characteristics of HDFS

- **Highly Scalable**: Designed to scale to thousands of nodes and petabytes of data.
- **Commodity Hardware**: Optimized for clusters composed of inexpensive, off-the-shelf machines.
- **Data Blocks**: Data files are broken into configurable blocks (default is 128 MB) and stored across DataNodes.
- **Replication**: Blocks are replicated (default 3x) across different DataNodes to improve fault tolerance and parallel read performance.
- **NameNode**: A centralized component (manager) that stores the metadata for the file system, including the mapping of data blocks to DataNodes/IPs.
- **Directory Structure**: A distributed file on HDFS appears as a directory containing individual filenames for each data block and metadata files.

Pros and Cons of HDFS

- **Pros**: Highly scalable, fault-tolerant, and designed for large-scale data processing.
- **Cons**: Primarily supports read-only access and batch-append operations. It does not offer efficient fine-grained updates or writes to existing data blocks.

Data-Parallel Dataflow / MapReduce

MapReduce is a seminal data-parallel dataflow framework for processing large datasets across clusters, originally introduced by Google.

Phases of MapReduce

The MapReduce model consists of two main phases:

1. **Map Phase**:
   - Input data is read from HDFS (or another distributed file system).
   - A user-defined ‘Map‘ function processes each input record (Key-Value pair) independently.
   - The ‘Map‘ function emits zero, one, or multiple intermediate Key-Value pairs.
2. **Reduce Phase**:
   - Intermediate Key-Value pairs are shuffled and grouped by key.
   - A user-defined ‘Reduce‘ function processes the list of values associated with each unique key.
   - The ‘Reduce‘ function produces zero, one, or more output Key-Value pairs, which are then written back to HDFS.

Characteristics of MapReduce

- **Fault Tolerance**: Achieved through re-execution of failed tasks and replication of intermediate data.
- **Scalability**: Designed for linear scalability with increasing cluster size.
- **Programming Model**: A simplified programming model that abstracts away distributed system complexities for the user.

Iterative Dataflow / Spark and Iterative MapReduce

The original MapReduce was not efficient for iterative algorithms (e.g., machine learning algorithms that run multiple passes over the data). This led to the development of systems like Apache Spark, which optimize for iterative dataflows by keeping intermediate data in memory across iterations, reducing I/O overhead. Iterative MapReduce frameworks also emerged to address this limitation.