Architectures

Arnav Gupta

December 11, 2024

Contents

1 Streaming vs Batch Processing

1.1 Scalability Problems

1.1.1 Easy

Handle independent requests from millions of customers.

Assume data is already available to service requests.

Trivially parallelizable, use horizontal scaling to divide and conquer.

1.1.2 Hard

Perform calculation use all data from millions of customers.

Requires lots of coordination, housing data on compute notes.

Must use MapReduce, Spark, etc.

1.2 Big Data

Companies have exponential growth of data, so big data critical for organizations and driven by technological advancements.

Big Data processing cycle include collection, preparation, input, processing, output/interpretation, and storage.

Data analytics essential for extracting valuable insights and support decision-making and strategy.

Cloud computing facilitates data storage, processing, and analysis from diverse data sources.

For Big Data problems best to split analysis into subproblems and parallelize over machines.

1.3 Distributed Computing

Data and analysis distributed across machines.

No machine has access to all data. Machines must communicate over network to share data and intermediate results.

Some calculations like sum, max, min, mean, search/grep can be easy to parallelize, with most work done with no coordination and little communication.

Input data split among nodes. Code distributed to many connected nodes. Many rounds of communication may be needed for a successful distributed algorithm.

Distributed algorithm can be written from scratch in any language by reading/writing memory and files and making network connections to other nodes to communicate.

Distributed coding frameworks manage common tasks required by Big Data analyses: MapReduce/Hadoop, Dryad, Spark, MPI, OpenMP.

1.3.1 Hadoop

Computing cluster for storing data and running analyses.

Components are:

- HDFS (Hadoop Distributed File System): for storing data to be analyzed and for storing intermediate and final results
  - data distributed across all nodes (machines) in the cluster
  - HDFS closely coupled to analysis engine
  - moving/querying data too slow, so run analysis on storage cluster
- MapReduce: programming model for defining parallel analysis steps
  - usually written in Java
  - defined in terms of Map and Reduce
    - Map does something, producing intermediate results
    - Reduce aggregates intermediate results
  - framework handles distribution of intermediate results and job scheduling
  - programmer must define map and reduce functions to complete work in parallel
    - map must take a key and value and return some list of key-value pairs (set of intermediate pairs)
    - reduce combines all intermediate values for a particular key and produces a set of merged output values (usually just one)

Input splitting and shuffling steps are done automatically by the Hadoop runtime system.

Inter-node communication handled entirely by shuffle step and distributed file system splitting data across compute nodes. Input files stored on each node are assigned to that node's mapper, reducing communication overhead.

HDFS distributes input and output data among many nodes.

Runtime system manages job scheduling and coordination.

Hadoop has scalable, distributed sorting algorithm for shuffling step.

1.4 Batch Processing

Processing large data chunks for data stored over time. Have jobs tart and finish, and process in sequential order.

Advantages include efficiency for large jobs and working offline and conserving resources.

Batch processing is useful for transactions and orders, has high latency, so better for non-time-sensitive jobs, and can be done with Hadoop.

Batch processing is crucial for structured, large-volume data.

Lifecycle:

- data collection and preparation
- data storage and analysis
- result generation and interpretation

Challenges include:

- managing large data volumes
- ensuring data quality and security

1.5 Stream Processing

Real-time data processing, so low-latency and continuous data flow.

Gives immediate insights for rapid decision making.

Better for high-velocity data streams, and can be used for fraud detection, recommendations, and monitoring systems.

Challenges include:

- data volume and velocity
- real-time analysis requirements
  - immediate data processing and analysis
- resource management
  - efficient use of memory and CPU

1.5.1 Apache Storm

Real-time computation system for data stream.

Components are:

- spout: data stream source
- bolt: data processing unit

Provides scalability and fault tolerance.

Used for real-time analytics and remote procedure calls.

Easy to setup and operate and integrates with various data sources.

Has complex state management and limited data windowing capabilities.

1.6 Batch vs Stream

Batch for large data volumes, stream for real-time analytics (low-latency)..

Batch processing uses MapReduce for offline analysis, with scheduler-based job execution.

Stream processing uses stream processors that run online, continuous analysis, with continuous data processing suited for real-time decision making.

2 Leader-Follower Architecture

Concurrency pattern for distributed systems.

Core concept is dynamic role assignment among threads.

Leader handles events, followers await their turns.

Components:

- thread management
  - single active leader thread
  - pool of passive follower threads
- event handling
  - leader demultiplexes and dispatches events

Used for high-volume transaction systems, resource-constrained environments, and complex event processing.

Advantages:

- resource efficiency
  - reduced thread count and lowered system overhead
- scalability
  - handles fluctuating workloads effectively
- flexibility
  - dynamic role switching enhances adaptability
- simplified synchronization
  - reduced complexity in thread coordination
- improved throughput
  - efficient event handling and processing
- fault tolerance
  - leader failure leads to new leader election

Disadvantages:

- complexity in design and maintenance
  - careful coordination
- bottlenecks
  - leader thread
- debugging and monitoring
  - tracking issues in role assignments
- event consistency and ordering
  - ensure reliable event handling
- failure recovery mechanism
  - handle leader thread failure
- integration with legacy systems

- adapt pattern to existing infrastructures

Can have different leader election mechanisms and different approaches to thread synchronization.

Can use tools for coordinating with external systems.