DSC 232R: Big Data Analytics Using Spark
Winter 2026
Week 1

January 8, 2026

1 Topic: Introduction to Big Data Using Spark

1.1 What is Data Science

1.1.1 Lecture Content

- **Definition:** Data Science is defined as *rational decision making* using data to make decisions that are useful and profitable. It combines "Data" (collection from sensors, logs, etc.) and "Science" (hypothesis testing and verification).
- **Types of Decisions:**
  - **Big Decisions:** Strategic, high-stakes decisions involving deliberation by people.
    - *Examples:* Is city water safe?, Federal interest rate hikes, Infrastructure changes (adding lanes).
  - **Small Decisions:** Tactical, automated, high-volume decisions made without human intervention.
    - *Examples:* Ad selection (Google), Movie recommendations (Netflix), Loan approvals, Ramp metering (traffic lights on highway on-ramps).
- **Ingredients of Data Science:**
  1. **Math:** Linear Algebra, Probability, Statistics.
  2. **Machine Learning:** Algorithms to build flexible models.
  3. **Software Development:** Implementing at scale.
  4. **Domain Knowledge:** Understanding the specific science of the problem (e.g., Traffic Flow theory).

- **Case Study: Caltrans PeMS (Performance Measurement System):**
  - **Data Collection:** 45,000 magnetic loop detectors in CA highways.
  - **Measurements:**
    - *Flow:* Number of cars per unit time.
    - *Occupancy:* Fraction of time a car is over the loop.
  - **Traffic Theory (The Fundamental Diagram):** Relationship between *Density* (cars/mile) and *Flow*.
    - Low Density → High Speed, increasing Flow.
    - Peak Flow → Optimal capacity.
    - High Density (Traffic Jam) → Low Speed, decreasing Flow.
  - **Analysis Techniques:** Uses PCA (Principal Component Analysis) to identify traffic profiles (e.g., AM vs PM peaks).

1.2 Data Engineering and Data Science

1.2.1 Lecture Content

- **Roles:**
  - **Data Scientist:** Builds models, answers business questions, uses Stats/ML. Expects data availability and fast computation.
  - **Data Engineer:** Builds the infrastructure (databases, streaming, cloud, pipelines) that supports the data scientist.
- **Course Scope (Data Engineering Topics):**
  - **Covered:** Hadoop File System (HDFS), Data Partitioning, Caching/Persistence, Checkpointing.
  - **Not Covered:** Data Cleaning, Spark Server Optimization, Containerization.
- **Data Models (The Interface):**
  1. **Matrix (Linear Algebra):**
     - Rectangle of numbers (all same type).
     - Supports transposition, addition, multiplication.
     - *Limitation:* Typically must fit in the memory of **one** computer.
  2. **Relation/Table (Relational Databases):**
     - Rows = Tuples (Entities), Columns = Properties (Attributes).
     - Columns have types, but types can differ across columns.
     - *Scale:* Can span many disks/computers; uses indices for fast retrieval without loading everything into memory.

3. **DataFrame (The Hybrid):**
   - Blend of Matrix and Table concepts (popularized by R/S/Pandas).
   - Ordered, named rows and columns.
   - **Spark DataFrames:** Designed to reside on disk and support distributed processing (unlike standard in-memory matrices).

1.3 Speeding Up Data Processing

1.3.1 Lecture Content

- **Two Primary Optimization Methods:**
  1. **Fast Libraries (Vectorization):** Replacing explicit Python loops with calls to optimized libraries (e.g., NumPy).
  2. **Throughput vs. Latency:** Optimizing for volume rather than individual speed.
- **Throughput vs. Latency Definitions:**
  - **Latency:** Time to process a single item from start to finish.
    - *Analogy:* Time spent waiting in a grocery line.
    - *Parallelism:* Does NOT improve latency (adding 100 cashiers doesn't make *your* checkout faster).
    - *Critical for:* Gaming, High-Frequency Trading.
  - **Throughput:** Amount of data processed per unit time (e.g., bytes/sec).
    - *Analogy:* Total customers exiting the store per hour.
    - *Parallelism:* DOES improve throughput (100 cashiers process 100x more people).
    - *Critical for:* Data Science and Big Data processing.
- **The Big Data Bottleneck:**
  - Bottleneck is usually **Disk I/O** (Moving data Disk $\rightarrow$ Memory), not CPU speed.
  - *Example:* Processing 1TB of data.
    - Single Machine (200MB/s): $\sim$1.4 hours.
    - 100 Machines (Parallel): $\sim$50 seconds.
  - **Solution:** MapReduce/Spark organize this parallel processing on unreliable clusters.

1.3.2 Juypter Notebook Content: Numpy vs Pure Python

- **The Experiment:** Matrix Multiplication ($A \times B$) of size 100×100.
- **Results:**
  - **NumPy:** $\sim$0.4 ms (uses optimized C/Fortran libraries).
  - **Pure Python:** $\sim$500 ms (uses nested loops).
  - **Speedup:** NumPy is $\sim$1000x faster for this size.
- **Under the Hood:**
  - NumPy relies on **LAPACK** (Linear Algebra PACKage).
  - Written in **Fortran** (released 1992).
  - Highly optimized for vector/matrix operations.
- **Scaling Behavior:**
  - **Small Matrices ($< 10 \times 10$):** No significant advantage (overhead dominates).
  - **Large Matrices (300×300):** NumPy is $\sim$10,000x faster.

2 Topic: Memory Hierarchy

2.1 Latency Throughput and Memory Hierarchy

2.1.1 Lecture Content

- **Definitions:**
  - **Latency:** The total time to process one single unit from start to finish. (Analogy: Time waiting in line + checkout).
  - **Throughput:** The number of units processed per unit of time. (Analogy: Customers exiting the store per hour).
  - **Key Insight:** Throughput is **not** necessarily 1 Latency.
    - *Example:* A store with no lines has low latency (fast checkout), but if nobody visits, throughput is near zero.
- **Analogy: Costco (Wholesale) vs. Retail:**
  - **Wholesale (Batch Processing):** High Latency (2 hours to drive truck), but massive Throughput (transferring 1000s of bottles). This is the Big Data approach.
  - **Retail (Random Access):** Low Latency (30 seconds to grab a bottle), but low Throughput (one bottle at a time). This is the Interactive approach.
- **Hard Disk Mechanics:**
  - **Seek Time (Latency):** $\sim$10ms to physically move the read head.
  - **Sequential Read (Throughput):** $\sim$100 MB/s once the head is in position.
  - **The Trap of Random Access:** Reading 1 byte randomly requires the full 10ms seek time.
    - Result: Throughput drops to $\sim$100 Bytes/sec.
    - Solution: Always read in **blocks** (Sequential Access) to amortize the seek cost.
- **Data Transfer at Scale (AWS Snowball):**
  - Transferring 50TB over a 100Mbps university line takes $\sim$46 days.
  - Transferring 50TB via **AWS Snowball** (physical shipping via FedEx) takes $\sim$24 hours.
  - **Lesson:** For massive data, physical transport (high latency) offers superior throughput compared to network transfer.

2.2 Storage Latency

2.2.1 Lecture Content

- **The Basic Operation ($C = A \times B$):**
  - Requires 4 distinct steps, each adding to total latency:
    1. Read A from Storage (High Latency).
    2. Read B from Storage (High Latency).
    3. Compute $A \times B$ in CPU (Low Latency).
    4. Write C to Storage (High Latency).
- **The Bottleneck:**
  - In Big Data analysis, the majority of execution time is spent on **Steps 1, 2, and 4** (Storage I/O).
  - The actual computation (Step 3) is negligible compared to data movement.
- **Storage Hierarchy:**
  - Different storage types offer different trade-offs between Latency, Capacity, and Price.
  - **Low Latency:** Main Memory (RAM).
  - **Medium Latency:** Spinning Disk.
  - **High Latency:** Remote Computer / Cloud Storage.
- **Goal of Big Data Systems:** Organize storage and computation to maximize Throughput (processing speed) while minimizing Cost.

2.3 Memory Hierarchy

2.3.1 Lecture Content

2.3.2 Lecture Content

- **The Hierarchy Levels:**
  1. **CPU Registers:** Fastest ($\sim$ 300 ps), Smallest ($\sim$ 1 KB).
  2. **L1/L2/L3 Cache:** Fast ($\sim 1 - 20$ ns), Small ($\sim$ 64 KB - 4 MB).
  3. **Main Memory (RAM):** Moderate ($\sim$ 100 ns), Moderate Size ($\sim$ 16 GB).
  4. **Disk Storage:** Slow ($\sim$ 10 ms), Huge Size ($\sim$ 16 TB). *Note: 6 orders of magnitude slower than CPU!*
  5. **Network (Cluster):** Slowest, Massive Scale (10+ PB).
- **The Abstraction:**

  - Hardware presents memory as a single, large, flat array.
  - Performance relies on **Locality**: The hardware automatically moves frequently accessed data to the faster levels.

- **Cluster Computing (Spark Context):**
  - Extends the hierarchy via **Ethernet**.
  - **Locality Rule:** Compute data on the node where it resides to avoid network latency.
  - **Shuffling:** The cluster equivalent of moving data between levels (expensive).
  - **Spark RDD:** The abstraction that makes a cluster look like a single computer's memory.

2.4 Heavy Tail Distributions

2.4.1 Lecture Content

- **The Problem:** Memory is 1-Dimensional (linear addresses), but Matrices are 2-Dimensional. We must flatten 2D data into 1D storage.
- **Two Standards:**
  1. **Row-Major Order:** Consecutive elements of a **row** are stored together.
     - *Used by:* C, C++, Python, **NumPy**.
     - *Traversal:* Fast to iterate row-by-row (inner loop on columns).
  2. **Column-Major Order:** Consecutive elements of a **column** are stored together.
     - *Used by:* Fortran, MATLAB, R, Spark (often, due to Scala/JVM).
     - *Traversal:* Fast to iterate column-by-column (inner loop on rows).
- **Performance Impact:** Traversing a Row-Major array in Column order (or vice versa) breaks **Spatial Locality**.
  - The CPU fetches a cache line, uses 1 value, and discards the rest.
  - Result: Massive increase in Cache Misses and execution time.

2.4.2 Juypter Notebook Content: Row vs Col Major

- **The Experiment:** Summing elements of a $10k \times 10k$ NumPy matrix.
- **Results:**
  - **Row Traversal (Good Locality):** $\sim$8 seconds.
  - **Column Traversal (Bad Locality):** $\sim$100-200 seconds.

- **Factor:** 10x - 20x slowdown purely due to memory access patterns.

- **Key Takeaway:** Always match your iteration order to the language's storage layout. For NumPy, iterate over rows first.

2.4.3 Juypter Notebook Content: Measuring Performance of Memory Hierarchy

- **The Experiment:** Accessing random indices in arrays of increasing size (1KB to 1GB).

- **The Staircase Graph:**

  - **Step 1 (L1 Cache):** Sizes < 32KB. Latency ~ 0.5 ns.
  - **Step 2 (L2 Cache):** Sizes 32KB - 256KB. Latency $\sim 2 - 5$ ns.
  - **Step 3 (L3 Cache):** Sizes 256KB - 6MB. Latency ~ 10 ns.
  - **Step 4 (Main Memory):** Sizes $>$ 10MB. Latency ~ 100 ns.

- **Conclusion:** You can physically "see" the cache sizes of a computer by measuring access latency spikes.

2.4.4 Juypter Notebook Content: A More Accurate Measure of Memory Poke Latency

- **The Measurement Problem:** A naive loop 'for i in range(n): x = A[i]' measures both the memory access AND the loop overhead (Python logic, index increment).

- **The Solution (Loop Unrolling):**

  - Perform many accesses inside a single loop iteration.
  - Code: 'sum += A[i] + A[i+1] + ... + A[i+9]'

- **Effect:** Amortizes the loop overhead across many operations, bringing the measured time closer to the true hardware latency (approx 1 ns for L1 vs 3-4 ns naive).

3 Exam Traps: Danger Zone

- **The Definition Trap:** Data Science is not just "analyzing data"; the professor explicitly defines it as *rational decision making* (Big vs. Small decisions).
- **The Traffic Graph Trap:** Higher density does **not** always mean higher flow. After the "critical density" peak, increasing density *decreases* flow (Traffic Jam).
- **The "Matrix" Trap:** Matrices are homogeneous (numbers only) and must fit in **one** computer's memory. If the data is 50TB or has mixed types (strings/ints), it's a Table or DataFrame, not a Matrix.
- **The Parallelism Limit:** Parallelism improves **Throughput** (total volume), but it rarely improves **Latency** (time for one unit). Adding 100 computers won't make a single network request 100x faster.
- **The "Dirty" Penalty:** A cache miss is bad. A cache miss on a **dirty** block is worse because you pay double the latency: 1) Write the old dirty data to RAM, 2) Read the new data from RAM.
- **Language Defaults:** NumPy is **Row-Major**. Iterating column-by-column kills performance (10-20x slower) because it breaks spatial locality.
- **The Bottleneck Reality:** In Big Data, the bottleneck is almost always **Disk I/O** (moving data), not CPU speed. "Faster Math" doesn't help if the CPU is waiting for data.

4 Practice Quiz

Question 1

The Spatial Locality Check

Which operation benefits most from **Spatial Locality**?

a Updating a single global sum variable 1,000 times.

b Reading every element of a 10,000-element integer array.

c Traversing a binary tree with 10,000 nodes allocated randomly in memory.

d Calculating a factorial using recursion.

Answer: B

Brief Explanation

- **Spatial Locality** refers to accessing memory addresses that are *contiguous* (next to each other).
- An array is stored contiguously in memory. Loading the first element likely loads the next 64 bytes (Cache Line) automatically, making subsequent reads nearly instant.
- Option A is *Temporal* Locality (same address). Option C has poor locality (random pointers).

Question 2

Traffic Flow Theory

According to the Fundamental Diagram of Traffic, what happens when density exceeds the critical peak?

a Flow remains constant at maximum capacity.

b Flow increases linearly with density.

c Flow decreases as velocity drops significantly.

d Throughput increases, but Latency decreases.

Answer: C

Brief Explanation

- Beyond the peak density, cars are too close to move fast. The drop in speed outweighs the increase in density, causing a traffic jam where flow (throughput) drops.

Question 3

Data Models

Which Data Model is characterized by having named columns, heterogeneous types (different types for different columns), and is designed to scale across many disks?

a Matrix

b Relation (Table)

c Vector

d Tensor

Answer: B

Brief Explanation

- Matrices are homogeneous and memory-bound. Tables (Relations) allow different types per column and use indices to scale to massive sizes on disk.

Question 4

Latency vs. Throughput

You need to transfer 50TB of data. Option A (Fiber Optic) takes 11 hours. Option B (AWS Snowball/FedEx) takes 24 hours. Which statement is true?

a Option A has higher Latency and higher Throughput.

b Option B has higher Latency, but could have higher Throughput if data size increases to 100TB.

c Option A is always preferred because Latency is lower.

d Throughput is identical because the data size is the same.

Answer: B

Brief Explanation

- Physical transport (Snowball) has a fixed high latency (24h travel time) but massive bandwidth. If you scaled to 100TB, Fiber would take 22 hours, while the truck still takes 24 hours, making the throughput comparison dependent on volume.

Question 5

4.0.1 NumPy Speed

Why is NumPy matrix multiplication (∼1ms) vastly faster than a pure Python implementation (∼500ms) for a 100×100 matrix?

a NumPy uses GPU acceleration automatically.

b NumPy uses a specialized C++ compiler at runtime.

c NumPy calls optimized Fortran libraries (LAPACK).

d NumPy avoids memory storage by streaming data.

Answer: C

Brief Explanation

- NumPy relies on BLAS/LAPACK libraries, originally written in Fortran (1992), which are highly optimized for vector operations.

Question 6

Cache Miss Penalty

In the context of the Memory Hierarchy, what is a "Dirty Eviction"?

a Removing data that has been corrupted.

b Removing data that was read but never used.

c Evicting a cache block that has been modified, requiring a write-back to RAM.

d Clearing the cache to prevent security leaks.

Answer: C

Brief Explanation

- If the CPU modified a value in the cache ("dirty"), that value must be saved to main memory before the cache slot can be reused. This doubles the latency penalty.

Question 7

Row vs. Column Major

You are iterating through a standard NumPy array using a nested loop. To maximize performance, which index should be in the **inner** loop?

a The Row index (iterate top-down).

b The Column index (iterate left-right).

c It does not matter for NumPy.

d It depends on the size of the array.

Answer: B

Brief Explanation

- NumPy is **Row-Major**. Elements in the same row are stored next to each other. Iterating along the column index (moving left-to-right across a row) accesses contiguous memory, maximizing spatial locality.

Question 8

Hierarchy Scale

Approximately how much slower is a Random Disk Access compared to a Main Memory (RAM) Access?

a 10x

b 100x

c 1,000x

d 100,000x (5 orders of magnitude)

Answer: D

Brief Explanation

- RAM access is $\sim$100 nanoseconds. Disk seek is $\sim$10 milliseconds. $10ms = 10,000,000ns$. The difference is roughly 5-6 orders of magnitude (10^5).

Question 9

Big Data Latency Chain

When calculating $C = A \times B$ on a dataset larger than memory, which step dominates the execution time?

a Computing the product in the ALU.

b Reading A and B from Disk.

c Allocating memory variables.

d The operating system scheduler.

Answer: B

Brief Explanation

- In Big Data, the bottleneck is moving data from slow storage (Disk) to fast memory. The CPU spends most of its time waiting for data (I/O Bound).

Question 10

Data Structures

Why does a Linked List generally exhibit poor performance compared to an Array for large sequential scans?

a Linked Lists have $O(N^2)$ access time.

b Linked Lists use more memory for integers.

c Linked List nodes are scattered in memory, causing frequent Cache Misses.

d Arrays are always cached in L1 by default.

Answer: C

Brief Explanation

- Linked list nodes are allocated dynamically and linked via pointers, scattering them across memory pages. Traversing them requires loading many different pages, whereas an Array loads many elements in a single page (Spatial Locality).