Project: A Distributed Engine for Large-Scale Sparse Matrix and Tensor Algebra

Amir Noohi

Programming for Data Science at Scale (PDSS)

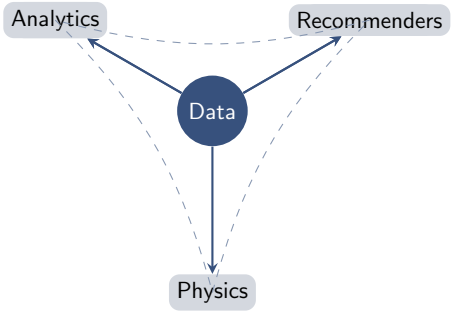
2025–2026

The Heartbeat of Modern Data Science

Matrix and tensor operations are the fundamental building blocks of countless algorithms. At scale, their efficient execution is critical.

Applications:

- PageRank (Web-scale graph analysis)
- Recommendation Systems (e.g., Netflix)
- Deep Learning (Neural network training)
- Scientific Computing (Simulations)



The Challenge

How do we multiply matrices with billions of entries that won't fit on one machine? Especially when they are **sparse**?

Your Mission

Your task is to design and implement a **distributed engine** for large-scale sparse matrix and tensor operations using Apache Spark.

Core Goal: Implement the following fundamental operations efficiently in distributed environment.

- **Sparse Matrix-Vector Multiplication (SpMV):** $y = A \cdot x$
- **Sparse Matrix-Matrix Multiplication (SpMM):** $C = A \cdot B$
- **Tensor Algebra**

You will need to think about the entire pipeline, from the user interface to the low-level data layouts and execution strategies.

Defining Your Project Scope

You must implement a baseline and build your optimizations upon it. Please specify which of the following combinations you will implement in your report.

Table: Select the combinations you will implement.

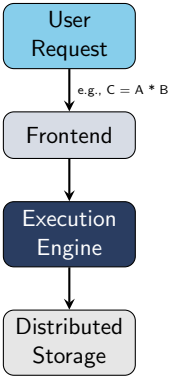
Operation	**Left Operand**	**Right Operand**	**Implemented? (✓)**
SpMV	Sparse Matrix	Dense Vector	
SpMV	Sparse Matrix	Sparse Vector	
SpMM	Sparse Matrix	Dense Matrix	
SpMM	Sparse Matrix	Sparse Matrix	
Tensor Algebra	(e.g., MTTKRP)		

The Components of Your Engine

We can break the system down into two major components.

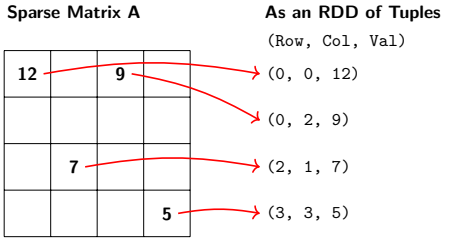
1. **Frontend (10%)**
   - User-facing API.
   - How does a user specify an operation?
   - Handles different operand types (Sparse & Dense).

2. **Execution Engine (35%)**
   - The "brain" of the system.
   - Manages data representation.
   - Implements the distributed algorithm.



How Do You Represent Data in a Distributed World?

A matrix is a 2D structure, but RDDs are a flat collection of records. How do you bridge this gap?



Your choice of data layout is the **single most important decision** you will make.

Data Layout Breakdown (10%):

- Loading mechanisms: 2%
- Dense representations: 4% (2% matrix, 2% vector)
- Sparse representations: 4% (2% matrix, 2% vector)

Implementing the Multiplication Logic

You must implement the core logic using Spark's distributed data-parallel operations.

- **Core Idea:** Decompose the distributed tensor operation into many small, independent calculations that can be run in parallel across multiple workers.
- **Your Toolkit:** RDD operations like 'map', 'flatMap', 'groupByKey', and especially **'join'**.

Warning!

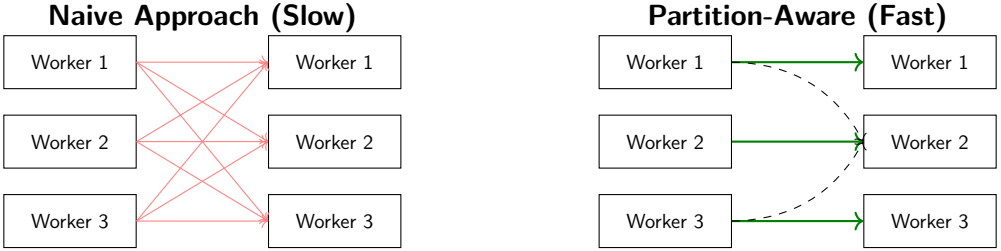
The use of 'collect()' on any large RDD to bring data to the driver for computation is strictly forbidden. The computation **must** remain parallelized across cores!

Runtime Engine Requirements (15%):

- Implement each operation in terms of RDD operations ($4 \times 3\% = 12\%$)
- Ensure using joins and no unnecessary collect (3%)

Distributed Optimizations: Reducing Data Movement & Memory Pressure

A naive implementation will create excessive data shuffling between workers. The solution is intelligent **Data Partitioning**.



Key Techniques (10%):

- Hashing strategies for optimal data distribution
- Partitioning schemes to minimize data movement

Advanced Techniques for Getting Full Marks

Once you have a working, optimized engine, explore more advanced techniques. These are excellent topics for the "Further Efficiency" section of your report.

Advanced Data Layout (5%)

The COO format is simple but not always the most efficient. Research and consider layouts like:

- **CSR, DCSR:** Very efficient for SpMV.
- **CSF:** If implementing tensor algebra.

Algebraic Optimizations (5%)

Can you use mathematical properties to reduce the amount of computation or communication?

- For example, in a chain of multiplications ($A \cdot B \cdot C$), the order matters!
- Other algabraic optimization like distributivity law ($A \cdot (B + C) = A \cdot B + A \cdot C$).

If you can't measure it, you can't improve it!

A huge part of this project is to **scientifically prove** that your design choices and optimizations are effective. Your evaluation must be rigorous.

- **Micro benchmarks (5%):**
  - Performance analysis against DataFrame
- **Impact of Distributed Optimization (10%):**
  - Measure effectiveness of your distributed optimizations
- **Further Ablation Study (5%):**
  - Advanced Data Layout: 2.5%
  - Algebraic Optimization: 2.5%
- **End-to-end evaluation (10%):**
  - Complete system performance evaluation
- **Another competitor (5%):**
  - Compare against alternative real-world implementations (beyond a single tensor operator)
- **Real-world application (5%):**
  - Demonstrate with practical use cases (beyond a single tensor operator)

Two-Part Coursework System

This project consists of **two interconnected courseworks**:

Coursework 1 (70%)

Group Project Implementation

- Groups of 3 students
- Implement distributed engine
- All group members submit **identical files**
- Released: 7 Oct (Tue noon)
- **Deadline: 14 Nov (Fri noon)**
- **Feedback by: 5 Dec 2025**

Coursework 2 (30%)

Peer Review Process

- Individual peer review assignment
- Review another group's submission
- 4 sections: System Design, Backend, Evaluation, Discussion
- Minimum word counts & detailed feedback required
- Released: 17 Nov (Mon noon)
- **Deadline: 28 Nov (Fri noon)**
- **Feedback by: 12 Dec 2025**

Important Submission Policies

CW1 - No Extensions

Rule 2: No Extensions and no ETAs permitted because it is group work. Late individual submissions score zero. Penalties applied to late group work submissions.

CW2 - Extensions Allowed

Rule 1: Extensions (4 days) and ETAs (7 days) permitted. Penalties applied to late submissions.

Key Difference: CW1 has strict no-extension policy, while CW2 allows extensions with penalties.

Complete Submission Requirements

Deliverables Checklist:

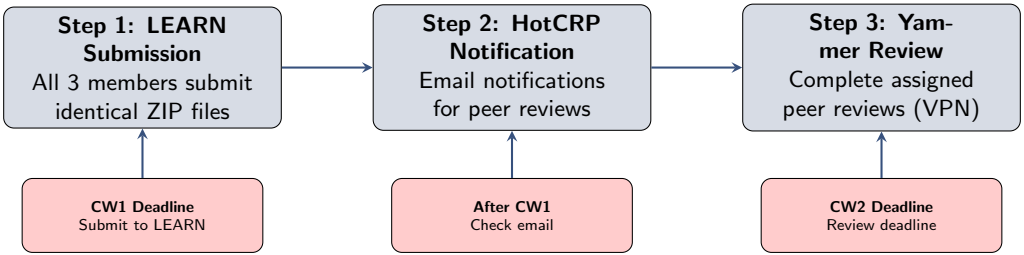
1. **Source Code** - Complete Scala/Spark implementation
2. **Report (PDF)** - 8-12 pages with results
3. **README.md** - Setup & running instructions
4. **Dependencies** - build.sbt configuration
5. **Test Data** - Sample datasets
6. **Results** - Benchmark graphs & analysis

Required File Structure:

`PDSS_Project_[GroupID]`

[DIR] `src/` → Scala source code
[DIR] `data/` → Sample datasets
[DIR] `results/` → Benchmarks
[FILE] `report.pdf` → Final report
[FILE] `README.md` → Instructions
[FILE] `build.sbt` → Dependencies

Complete Process Overview



ZIP Format: `PDSS_Project_[GroupID].zip`

Key Points

- **All students** must submit to LEARN
- **Check email** for HotCRP notifications
- **Yammer address:** `yammer.inf.ed.ac.uk`

CW2 - HotCRP Notifications & Yammer Reviews

VPN Required for Yammer Reviews

You MUST use School of Informatics VPN to access `yammer.inf.ed.ac.uk`

- Test VPN access **well before** the review deadline
- Contact Computing Support if you have VPN issues
- Complete your assigned reviews via Yammer platform

Honesty Declaration Required

LEARN submission includes honesty section:

- Declare how each group member contributed
- Report any group work issues **in advance**
- Contact staff if problems arise with group mates

What You Need to Review (CW2 - 30%)

You will review another group's report in **4 sections**:

System Design (25%)

3+ points (60+ words each)

- Strengths with examples
- Weaknesses with solutions

Backend & Optimizations (30%)

4+ points (60+ words each)

- Runtime, data layouts, optimizations
- Concrete improvement suggestions

Evaluation (30%)

3+ points (60+ words each)

- Assess benchmarks & ablation
- Suggest better strategies

Overall Discussion (15%)

1 summary (60+ words)

- Future work assessment
- Open challenges discussion

Key: All feedback must be **constructive** and **actionable**!

How Your Overall Grade is Calculated

Coursework 1 (70%)

- Introduction (10%)
- Frontend (10%)
- Execution Engine (35%)
- Further Efficiency (10%)
- Performance Evaluation (30%)
- Conclusion (5%)

Coursework 2 (30%)

Peer Review Quality:

- System Design (25%)
- Backend & Optimizations (30%)
- Evaluation (30%)
- Overall Discussion (15%)

Final Grade = (CW1 Grade $\times$ 0.7) + (CW2 Grade $\times$ 0.3)

Working in Groups & Review Process

Group Formation (CW1)

- **Groups of exactly 3 students**
- Group enrollment: 25 Sept (Thu noon) to 7 Oct (Mon noon)
- **Critical:** All 3 members must submit identical files

Peer Review Assignment (CW2)

- System randomly assigns reviews after CW1 deadline
- Each student reviews **one other group's** submission
- Reviews are **individual work** (anonymous process)

Important

You cannot review your own group's submission. The system ensures no conflicts of interest.

Support & Additional Resources

Discussion Forum: Piazza

- Ask technical questions
- Share (non-solution) resources
- Clarify project requirements

Where to Run Your Code:

- **Your laptop** - Local development and testing
- **Lab machines** - More cores and memory available
- **student.compute** - School of Informatics compute servers

Useful Resources:

- Apache Spark Scala API: `spark.apache.org/docs/latest/api/scala`
- SBT Documentation: `www.scala-sbt.org/documentation.html`

What's Allowed & What's Not

Academic Integrity Statement

All submitted work is expected to be your own. AI tools should not be used for this assessment.

Allowed

- Discussing high-level approaches and algorithms
- Sharing publicly available datasets and benchmarks

NOT Allowed

- Sharing or copying source code between students
- Submitting work that is not substantially your own
- Plagiarizing text in your report

When in doubt, ask! It's always better to clarify than to accidentally violate academic integrity policies.

Questions?