- Assignment 2 grades this week
- Midterm details on Piazza
- Course Project Proposal comments
Scalable Storage Systems

Datacenter Architecture

Resource Management

Computational Engines

Applications

- Machine Learning
- SQL
- Streaming
- Graph

- TensorFlow
- PyTorch
- Ray
- CFS
- Mesos
- DRF

- MapReduce
- Spark
- Gardiva
SQL: STRUCTURED QUERY LANGUAGE
DATABASE SYSTEMS

Queries

Data management layer

Admission Control

Dispatch and Scheduling

Local Client Protocols

Remote Client Protocols

Client Communications Manager

Query Parsing and Authorization

Query Rewrite

Query Optimizer

Plan Executor

Relational Query Processor (Section 4)

Access Methods

Buffer Manager

Lock Manager

Log Manager

Transactional Storage Manager (Sections 5 & 6)

Catalog Manager

Memory Manager

Administration, Monitoring & Utilities

Replication and Loading Services

Batch Utilities

Shared Components and Utilities (Section 7)
- Scale: How do we handle large datasets, clusters?

- Wide-area: How do we handle queries across datacenters?
Motivation: Understanding the structure of data

```scala
lines = sc.textFile("users")
csv = lines.map(x => x.split(','))
young = csv.filter(x => x(1) < 21)
println(young.count())
```
PROCEDURAL VS. RELATIONAL

ctx = new HiveContext()
users = ctx.table("users")
young = users.where(users("age") < 21)
println(young.count())

lines = sc.textFile("users")
csv = lines.map(x => x.split(',')).filter(x => x(1) < 21)
println(csv.count())
Projection (select), Filter, Join, Aggregations take in Expressions

employees.join(dept,
  employees ("deptId") === dept ("id ")
)

Build up Abstract Syntax Tree (AST)
OTHER FEATURES

1. Debugging: Eager analysis of logical plans
   \[ \text{user("address") = "101"} \]

2. Interoperability: Convert RDD to Dataframes
   - Relational: Optimizing, fewer code
   - Procedural: ETL - extract, transform, load
     RDD "Shivaram"
3. Caching: Columnar caching with compression

Parquet files from HDFS

4. UDFs: Python or Scala functions

```scala
val model: LogisticRegressionModel = ...

ctx.udf.register("predict", (x: Float, y: Float) => model.predict(Vector(x, y)))

ctx.sql("SELECT predict(age, weight) FROM users")
```
CATALYST

Goal: Extensibility to add new optimization rules
Library for representing trees and rules to manipulate them

tree. transform {
  case Add(Literal(c1),Literal(c2)) => Literal(c1+c2)
  case Add(left , Literal(0)) => left
  case Add(Literal(0), right) => right
}
1. Analyzer: Lookup relations, map named attributes, propagate types
2. Logical Optimization
   a. Predicate pushdown: reduce num rows that need to be processed
   b. Project pruning: reduce cols that need to be processed
3. Physical Planning
   - Join selection: Hash, Broadcast
   - Sort: merge
   - Pipeline filters, projections: map (Filter, Project)
CPU bound when data is in-memory
Branches, virtual function calls etc.

```python
def compile(node: Node): AST = node match {
    case Literal(value) => q"$value"
    case Attribute(name) => q"row.get($name)"
    case Add(left, right) =>
        q"${compile(left)} + ${compile(right)}"
}
```
EXTENSIONS

Data sources
- Define a BaseRelation that contains schema
- TableScan returns RDD[Row]
- Pruning / Filtering optimizations

User-Defined Types (UDTs)
- Support advanced analytics with e.g. Vector
- Users provide mapping from UDT to Catalyst Row
SUMMARY, TAKEAWAYS

Relational API
- Enables rich space of optimizations
- Easy to use, integration with Scala, Python

Catalyst Optimizer
- Extensible, rule-based optimizer
- Code generation for high-performance

Evolution of Spark API
DISCUSSION

https://forms.gle/r6DnV7wLGHjYmYd17
Does SparkSQL help ML workloads? Consider the MNIST code in your assignment. What parts of your code would benefit from SparkSQL and what parts would not?

1. Things not in ML workload
   - Pre-processing
   - ETL
   - Sampling

2. Batch operation on columns
   - Cache misses on row vs. column storage
   - Does it help compression

3. Eager analysis: Matrix sizes, indexing
1. SQL query not using column data
2. Query opt. Total data joined to drivers is smaller

SQL + Spark

DataFrame

Runtime (seconds)

- filter
- word count

2 stage pipeline

Moving data into FS

run more queries on DF
What are some limitations of the Catalyst optimizer as described in the paper? Describe one or two ideas to improve the optimizer.

2. Cool model

- General purpose fast on DR
- Improve Columnar Storage?
- Choose Storage format
NEXT STEPS

Next class: Wide-area SQL queries
Midterm coming up!
SCHEMA INFERENC

Common data formats: JSON, CSV, semi-structured data

JSON schema inference
- Find most specific SparkSQL type that matches instances
  e.g. if tweet.loc.latitude are all 32-bit then it is a INT
- Fall back to STRING if unknown
- Implemented using a reduce over trees of types