CS 744: SCOPE

Shivaram Venkataraman
Fall 2020
ADMINISTRIVIA

- Assignment grades this week
- Midterm details on Piazza
- Course Project Proposal Submission
Applications

Machine Learning | SQL | Streaming | Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture
SQL: STRUCTURED QUERY LANGUAGE
DATABASE SYSTEMS

Admission Control

Dispatch and Scheduling

Process Manager (Section 2)

Client Communications Manager

Local Client Protocols

Remote Client Protocols

Query Parsing and Authorization

Query Rewrite

Query Optimizer

Plan Executor

Relational Query Processor (Section 4)

DDL and Utility Processing

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

SELECT COUNT(*)
FROM "users"
WHERE age < 21
SELECT query, COUNT(*)
AS count
FROM "search.log"
USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;
Input reading: What is different?

EXTRACT column[::<type>][, ...]
FROM <input_stream(s)>
USING <Extractor> [(args)]
[HAVING <predicate>]
SQL OPERATORS

Select – read rows that satisfy some predicate
Join – Equijoin with support for Inner and Outer join
GroupBy – Group by some column
OrderBy – Sorting the output
Aggregations – COUNT, SUM, MAX etc.
R1 = SELECT A+C AS ac, B.Trim() AS B1 
FROM R 
WHERE StringOccurs(C,"xyz") > 2 

#CS
public static int StringOccurs(string str, string ptrn){
    int cnt=0; int pos=-1;
    while (pos+1 < str.Length) {
        pos = str.IndexOf(ptrn, pos+1);
        if (pos < 0) break;
        cnt++;
    }
    return cnt;
}
#ENDCS
MAPREDUCE-LIKE?

Process
Reduce
Combine

COMBINE S1 WITH S2
USING MultiSetDifference
PRODUCE A, B, C
SELECT query, COUNT() AS count
FROM "search.log"
USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;

Check syntax, resolve names
Checks if columns have been defined
Result: Internal parse tree
Rewrite the query expression \(\rightarrow\) lowest cost

Examples:
- Removing unnecessary columns
- Pushing down selection predicates
- Pre-aggregating

Also need to reason about partitioning
(See [VLDBJ paper](#))
RUNTIME OPTIMIZATIONS

Hierarchical aggregation

Locality-sensitive task placement

Grouping heuristics?
SUMMARY, TAKEAWAYS

Relational API
- Enables rich space of optimizations
- Easy to use, integration with C#

Scope Execution
- Compiler to check for errors, generate DAG
- Optimizer to accelerate queries (static + dynamic)

Precursor to systems like SparkSQL
DISCUSSION

https://forms.gle/hL8VJ6uSG7Lzm164A
Consider you have a column-oriented data layout on your storage system (Example below). What are some reasons that a SCOPE query might be faster than running equivalent MR program?

Row Storage

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>E-mail</th>
<th>Phone #</th>
<th>Street Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columnar Storage

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>E-mail</th>
<th>Phone #</th>
<th>Street Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Does SCOPE-like Optimizer help ML workloads? Consider the code in your Assignment2. What parts of your code would benefit and what parts would not?
NEXT STEPS

Next class: Elastic Data Warehousing with SnowFlake
Project proposals due tomorrow! See Piazza!
Midterm coming up!