Data Engineering

SQL Query Processing

Shivnath Babu
Spark SQL: Declarative Big Data Processing

Let Developers Create and Run Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work
Write Less Code: Compute an Average

Using RDDS

```scala
private IntWritable one = new IntWritable(1)
private IntWritable output = new IntWritable()
protected void map(
    LongWritable key, 
    Text value, 
    Context context)
{
    String[] fields = value.split("\t")
    output.set(Integer.parseInt(fields[1]))
    context.write(one, output)
}

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable()
protected void reduce(
    IntWritable key, 
    Iterable<IntWritable> values, 
    Context context)
{
    int sum = 0
    int count = 0
    for(IntWritable value : values) {
        sum += value.get()
        count++
    }
    average.set(sum / (double) count)
    context.write(key, average)
}
```

Using SQL

```
SELECT name, avg(age)
FROM people
GROUP BY name
```
Write Less Code: Compute an Average

Using RDDs

data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1]))
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]])
  .map(lambda x: [x[0], x[1][0] / x[1][1]])
  .collect()

Using SQL

SELECT name, avg(age)
FROM people
GROUP BY name
Spark Seamlessly Integrates SQL with a Full Programming Language

Embedding in a full programming language allows composition with functions to develop complex programs

```python
zipToCity = udf(lambda city: <custom logic here>)

def add_demographics(events):
    u = sqlCtx.table("users")
    events \
        .join(u, events.user_id == u.user_id) \
        .withColumn("city", zipToCity(df.zip))
```
def add_demographics(events):
    u = sqlCtx.table("users")  # Load partitioned table
    events \n        .join(u, events.user_id == u.user_id) \  # Join on user_id
        .withColumn("city", zipToCity(df.zip)) # Run func to add city column
    events = add_demographics(sqlCtx.load("/data/events", "parquet"))
    training_data = events.where(events.city == "Melbourne").select(events.timestamp).collect()
Plan Optimization & Execution

Diagram:
- SQL AST
- DataFrame
- Catalog
- Unresolved Logical Plan
- Logical Plan
- Optimized Logical Plan
- Physical Plans
- Cost Model
- Selected Physical Plan
- RDDs

Steps:
1. Analysis
2. Logical Optimization
3. Physical Planning
4. Code Generation
Query Processing

Declarative SQL Query → Query Plan
SQL Primer

SPJ, or Select-Project-Join Queries

Select <attribute list>
From <relation list>
Where <condition list>

Example Filter Query over R(A,B,C):
Select B
From R
Where R.A = “c” ∧ R.C > 10
SQL Primer (contd.)

SPJ, or Select-Project-Join-Queries

Select  <attribute list>
From    <relation list>
Where   <condition list>

Example Join Query over R(A,B,C) and S(C,D,E):
Select  B, D
From    R, S
Where    R.A = “c” ∧ S.E = 2 ∧ R.C = S.C
Select B,D
From R,S
Where R.A = “c” ∧
S.E = 2 ∧ R.C = S.C

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
<td></td>
<td>10</td>
<td>x</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>20</td>
<td></td>
<td>20</td>
<td>y</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
<td></td>
<td>30</td>
<td>z</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>35</td>
<td></td>
<td>40</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>45</td>
<td></td>
<td>50</td>
<td>y</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Answer

<table>
<thead>
<tr>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>x</td>
</tr>
</tbody>
</table>
• How do we execute this query?

Select B,D
From R,S
Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C

One idea
- Do Cartesian product
- Select tuples
- Do projection
<table>
<thead>
<tr>
<th></th>
<th>R.A</th>
<th>R.B</th>
<th>R.C</th>
<th>S.C</th>
<th>S.D</th>
<th>S.E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>y</td>
<td>2</td>
</tr>
</tbody>
</table>

Select B,D
From R,S
Where R.A = “c”
∧ S.E = 2 ∧
R.C=S.C

Bingo!
Got one...

<table>
<thead>
<tr>
<th></th>
<th>R.A</th>
<th>R.B</th>
<th>R.C</th>
<th>S.C</th>
<th>S.D</th>
<th>S.E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c</td>
<td>2</td>
<td>10</td>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
Relational Algebra - can be used to describe plans

Ex: Plan I

\[ \Pi_{B,D} (\sigma_{R.A = "c" \land S.E = 2 \land R.C = S.C} (R \times S)) \]
Relational Algebra Primer

Select: $\sigma_{R.A=\text{"c"}} \land R.C=10$
Project: $\Pi_{B,D}$
Cartesian Product: $R \times S$
Natural Join: $R \bowtie S$
Relational Algebra - can be used to describe plans

Ex: Plan I

\[ \Pi_{B,D} \]

\[ \sigma_{R.A = "c" \land S.E = 2 \land R.C = S.C} \]

\[ R \times S \]

OR: \[ \Pi_{B,D} [ \sigma_{R.A = "c" \land S.E = 2 \land R.C = S.C} (R \times S) ] \]
Another idea:

Plan II

\[ \Pi_{B,D} (\sigma_{R.A = "c"} R(A,B,C) \quad \sigma_{S.E = 2} S(C,D,E)) \]

Select B,D
From R,S
Where R.A = "c" \( \land \)
S.E = 2 \( \land \) R.C=S.C
Select B,D
From R,S

Where R.A = “c”  ∧
S.E = 2  ∧  R.C=S.C
Plan III

Use R.A and S.C Indexes

(1) Use R.A index to select R tuples with R.A = “c”

(2) For each R.C value found, use S.C index to find matching tuples

(3) Eliminate S tuples S.E ≠ 2

(4) Join matching R,S tuples, project B,D attributes, and place in result
\[
\text{A} = \text{"c"}
\]

\[
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
a & 1 & 10 \\
b & 1 & 20 \\
c & 2 & 10 \\
d & 2 & 35 \\
e & 3 & 45 \\
c & 7 & 15 \\
\hline
\end{array}
\]

\[
\text{output: } <2,x>
\]

\[
\text{next tuple: } <c,7,15>
\]
Overview of Query Processing

1. Parse
   - SQL query
   - Parse tree

2. Query rewriting
   - Logical query plan

3. Physical plan generation
   - Physical query plan

4. Execute
   - Result

Query Optimization

Query Execution
Example Query

Select B,D
From R,S
Where R.A = "c" \land R.C=S.C
Example: AST: Abstract Syntax Tree

```
SELECT  <SelList>    FROM    <FromList>     WHERE     <Cond>
          <Attribute> <SelList>   <RelName> <FromList> <Cond>  AND       <Cond>
B          <Attribute>        R         <RelName>        S
          <Attr> <Op> <Const>  <Attr> <Op> <Attr>
R.A  =  "c"                  R.C = S.C
```

Select B,D
From R,S
Where R.A = “c” ∧ R.C=S.C
Along with Parsing …

• Semantic checks
  – Do the projected attributes exist in the relations in the From clause?
  – Ambiguous attributes?
  – Type checking, ex: R.A > 17.5

• Expand views
Query rewriting

Initial logical plan

Rewrite rules

"Best" logical plan

Physical plan generation

execute

result
Initial Logical Plan

Relational Algebra: \( \Pi_{B,D} \left[ \sigma_{\text{R.A} = "c" \land \text{R.C} = \text{S.C}} \left( \text{RXS} \right) \right] \)
Apply Rewrite Rule (1)

\[ \pi_{B,D} [ \sigma_{R.C = S.C} [ \sigma_{R.A = "c"}(R \times S)]] \]
Apply Rewrite Rule (2)

\[ \Pi_{B,D} \left[ \sigma_{R.C = S.C} \left[ \sigma_{R.A = "c"}(R) \right] \right] \times S \]
Apply Rewrite Rule (3)

\[
\pi_{B,D} \left( \sigma_{R.A = "c"}(R) \right) \quad \text{Natural join} \quad \pi_{B,D} \left( \sigma_{R.C = S.C}(X) \right)
\]

\[
\Pi_{B,D} \left[ \sigma_{R.A = "c"}(R) \bowtie S \right]
\]
Some Query Rewrite Rules

• Transform one logical plan into another
  – Do not use statistics
• Equivalences in relational algebra
• Push-down predicates
• Do projects early
• Avoid cross-products if possible
Equivalences in Relational Algebra

\[ R \bowtie S = S \bowtie R \quad \text{Commutativity} \]
\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \quad \text{Associativity} \]

Also holds for: Cross Products, Union, Intersection

\[ R \times S = S \times R \]
\[ (R \times S) \times T = R \times (S \times T) \]
\[ R \cup S = S \cup R \]
\[ R \cup (S \cup T) = (R \cup S) \cup T \]
Apply Rewrite Rule (1)

\[ \pi_{B,D} \left[ \sigma_{R.A = "c" \land R.C = S.C} (R \times S) \right] \]

\[ \sigma_{R.C = S.C} \]

\[ \sigma_{R.A = "c"} \]

\[ \pi_{B,D} \]

\[ \Pi_{B,D} \left[ \sigma_{R.C = S.C} \left[ \sigma_{R.A = "c"}(R \times S) \right] \right] \]
Rules: Project

Let: \( X = \) set of attributes
\( Y = \) set of attributes
\( XY = X \cup Y \)

\( \pi_{xy}(R) = \pi_x[\pi_y(R)] \)
Let $p$ = predicate with only $R$ attribs
$q$ = predicate with only $S$ attribs
$m$ = predicate with only $R, S$ attribs

Rules: $\sigma + \bowtie$ combined

$\sigma_p (R \bowtie S) = \left[ \sigma_p (R) \right] \bowtie S$

$\sigma_q (R \bowtie S) = R \bowtie \left[ \sigma_q (S) \right]$
Rules: $\sigma + \bowtie$ combined (continued)

\[ \sigma_{p \land q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)] \]

\[ \sigma_{p \land q \land m} (R \bowtie S) = \]
\[ \sigma_m [(\sigma_p R) \bowtie (\sigma_q S)] \]

\[ \sigma_{p \lor q} (R \bowtie S) = \]
\[ [(\sigma_p R) \bowtie S] \cup [R \bowtie (\sigma_q S)] \]
Which are “good” transformations?

- $\sigma_{p1 \land p2} (R) \rightarrow \sigma_{p1} [\sigma_{p2} (R)]$
- $\sigma_{p} (R \bowtie S) \rightarrow [\sigma_{p} (R)] \bowtie S$
- $R \bowtie S \rightarrow S \bowtie R$
- $\pi_{x} [\sigma_{p} (R)] \rightarrow \pi_{x} \{\sigma_{p} [\pi_{xz} (R)]\}$
Conventional wisdom: do projects early

Example: $R(A,B,C,D,E)$

$P$: $(A=3) \land (B=\text{“cat”})$

$\pi_E \{ \sigma_p (R) \} \quad \text{vs.} \quad \pi_E \{ \sigma_p \{ \pi_{ABE} (R) \} \}$
But: What if we have A, B indexes?

B = “cat”

A=3

Intersect pointers to get pointers to matching tuples
Bottom line:

• No transformation is always good
• Some are usually good:
  – Push selections down
  – Avoid cross-products if possible
  – Subqueries → Joins
Avoid Cross Products (if possible)

Select B,D
From R,S,T,U
Where R.A = S.B ∧
R.C=T.C ∧ R.D = U.D

• Which join trees avoid cross-products?
• If you can't avoid cross products, perform them as late as possible
More Query Rewrite Rules

• Transform one logical plan into another
  – Do not use statistics
• Equivalences in relational algebra
• Push-down predicates
• Do projects early
• Avoid cross-products if possible
• Use left-deep trees
• Subqueries → Joins
• Use of constraints, e.g., uniqueness
Query rewriting

- Parse
  - Parse tree

Query rewriting

- Best logical query plan
  - Statistics

Physical plan generation

- Best physical query plan

Execute

- Result
Physical Plan Generation

π_{B,D}

σ_{R.A = "c"}

Best logical plan

Natural join

Hash join

Project

Index scan

Table scan
Query rewriting

- SQL query
  - parse
    - parse tree
  - Query rewriting
    - Best logical query plan
      - Enumerate possible physical plans
        - Find the cost of each plan
          - Pick plan with minimum cost
  - Physical plan generation
    - Best physical query plan
      - execute
        - result
Physical Plan Generation

Logical Query Plan

P1

P2

....

Pn

C1

C2

....

Cn

Pick minimum cost one

Physical plans

Costs
Operator Plumbing

\[ \pi_{B,D} \]
\[ \sigma_{R.A = "c"} \]
\[ S \]
\[ R \]

- **Materialization**: output of one operator written to disk, next operator reads from the disk
- **Pipelining**: output of one operator directly fed to next operator
Materialization

Materialized here

$\pi_{B,D}$

$\sigma_{R.A = "c"}$

S

R
Iterators: Pipelining

Each operator supports:
- Open()
- GetNext()
- Close()
Iterator for Table Scan (R)

Open() {
    /** initialize variables */
    b = first block of R;
    t = first tuple in block b;
}

GetNext() {
    IF (t is past last tuple in block b) {
        set b to next block;
        IF (there is no next block)
            /** no more tuples */
            RETURN EOT;
        ELSE t = first tuple in b;
    }
    /** return current tuple */
    oldt = t;
    set t to next tuple in block b;
    RETURN oldt;
}

Close() {
    /** nothing to be done */
}
Iterator for Select

\[ \sigma_{R.A = \text{"c"}} \]

**Open()**

```csharp
/** initialize child */
Child.Open();
```

**Close()**

```csharp
/** inform child */
Child.Close();
```

**GetNext()**

```csharp
LOOP:
t = Child.GetNext();
IF (t == EOT) {
    /** no more tuples */
    RETURN EOT;
}
ELSE IF (t.A == "c")
    RETURN t;
ENDLOOP:
```

**Example**

\[ \sigma_{R.A = \text{"c"}} \]
Iterator for Sort

\[ \tau_{R.A} \]

**Open()**

/* Bulk of the work is here */
Child.Open();
Read all tuples from Child and sort them

**GetNext()**

IF (more tuples)
    RETURN next tuple in order;
ELSE RETURN EOT;

**Close()**

/* inform child */
Child.Close();
Iterator for Tuple Nested Loop Join

• TNLJ (conceptually)
  for each \( r \in Lexp \) do
    for each \( s \in Rexp \) do
      if \( Lexp.C = Rexp.C \), output \( r,s \)
Example 1: Left-Deep Plan

Question: What is the sequence of getNext() calls?
Example 2: Right-Deep Plan

![Diagram showing a right-deep plan with operations and relations]

Question: What is the sequence of getNext() calls?
Cost Measure for a Physical Plan

• There are many cost measures
  – Time to completion
  – Number of I/Os (we will see a lot of this)
  – Number of getNext() calls

• Tradeoff: Simplicity of estimation Vs. Accurate estimation of performance as seen by user