Nian Ke
David R. Cheriton School of Computer Science
University of Waterloo

Advanced partitioning techniques for massively distributed computation
Outline

- Background
  - MapReduce Model
  - SCOPE Language and Cosmos system
- Advanced partitioning techniques
  - Partial Partitioning
    - Hash-Based Partitioning
    - Range-Based Partitioning
  - Indexed-based Partitioning
- Critiques and Discussion
Background

- MapReduce Model
- SCOPE Language and Cosmos system
MapReduce Model
Limitations of MapReduce

- Expertise are required to translate the application logic to MapReduce model in order to achieve parallelism.
- Code can be hard to debug and almost impossible to be reused.
- Complex application can become cumbersome to implement.
- Optimization of MapReduce jobs could be difficult.
SCOPE (Structured Computation Optimized for Parallel Execution) Language and Cosmos System

```sql
select R.c, S.d, count(*)
from R, S
where R.a = S.a and R.b = S.b and p1(R) and p2(S)
group by R.c, S.d
```

```
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
```

SCOPE Script

FrontEnd Services

SCOPE Compiler

SCOPE Runtime

SCOPE Optimizer

Execution Environment

Cosmos Storage System

Unstructured Streams

Structured Streams

SCOPE Layer

Data = SELECT * FROM S
WHERE Col1 > 10

SCOPE Compiler

SCOPE Optimizer

SCOPE Run/T

SCOPE Run/T

SCOPE Run/T
Advanced partitioning techniques

- Partial Partitioning
  - Hash-Based Partitioning
  - Range-Based Partitioning
  - Indexed-based Partitioning
Even after query optimization, certain repartitions are still inevitable.

However by carefully define the partition scheme, we could use partial repartitioning to replace full repartitioning.

Partial partitioning could greatly reduce I/O, communication and memory burden while relieve the scheduler and decrease response time.
If the input has already been hash partitioned by a, a great deal of resources would be saved.

```sql
SELECT a, UDAgg(b) AS aggB
FROM SSTREAM "input.ss"
GROUP BY a;

OUTPUT TO SSTREAM "output.ss"
    [HASH | RANGE] CLUSTERED BY a;
```
Example 1 Suppose that \( pi = 4 \) and \( po = 2 \) (i.e., we want to partition 2-ways an input that is already 4-way partitioned). Every row in \( P_0 \) satisfies \( h(C) \equiv 0 \mod 4 \), where \( h \) is the hash function and \( C \) are the partitioning columns. Figure 5(a) shows the default partitioning strategy which connects every input vertex with every output vertex. In this case, we know that \( h(C) \equiv 0 \mod 2 \) as well, and therefore \( P_0 \) would never generate a row satisfying \( h(C) \equiv 1 \mod 2 \). Thus, \( M_1 \) does not need to read the empty local partition produced by \( P_0 \). In general, \( M_0 \) only reads from \( P_0 \) and \( P_2 \), and \( M_1 \) from \( P_1 \) and \( P_3 \). Figure 5(b) shows the refined merge graph. A similar strategy can be applied when \( pi = 2 \) and \( po = 4 \). Figure 5(c) shows the refined partitioning graph.
Range-Based Partial Partitioning could be used when input and output partition scheme share common prefix.

Determine the partition boundary is important because it is crucial to reduce latency.
Algorithm 1: PartitionBoundaries(C, T, B)

Input: Columns C, Partition size T, Buckets B
Output: Partition boundaries P

/* Assume that C and B.cols share common prefix CP
and for each 1 < i < |B|: B[i-1].hi = B[i].lo */
/* Output is partitioned by CP, which implies C,
and each partition size is around T */
CB = \bigcup_i [\Pi_{CP}B[i].lo, \Pi_{CP}B[i].hi]  // project B on CP
idx = 0;
while idx < |CB| do
    actLo = CB[idx].Lo;
    actSize = CB[idx].Size;
    idx++;
    while actLo = CB[idx].Lo OR
        CB[idx].size / 2 < T - actSize do
            actSize += CB[idx].size;
            idx++;
    end
    P = P \cup [actLo, CB[idx-1].hi);
end
return P;
Range-Based Partial Partitioning

- Boundary decision could not only be made at compile time but also running time.
- Although extra cost is needed, it could avoid skewed partition in certain cases which would lead to high latency.

The StatCollector intercept the input and compute a histogram on the partitioning columns. Then the Coordinator compute a overall histogram and decide the overall partition boundaries.
Integrating the Techniques into SCOPE Optimizer

- Optimizer would eliminate certain repartition when certain functional dependency is detected between input partition scheme and potential output partition scheme.
- Optimizer chooses to repartition data based on requirements of subsequent operators.
- Optimizer would consider partial repartition if certain structural properties are detected. Compromise may also occur.
SCOPE Optimizer and Structured Streams

```
SELECT GetDomain(URL) AS Domain,
       SUM(MyNewScoreFunction(A, B, ...)) AS TotalScore
FROM Web-Table
GROUP BY Domain;

SELECT TOP 100 Domain ORDER BY TotalScore;
```

Super Expensive

Expensive

Unstructured Datasets

Structured Datasets (Sstream)
(partitioned by URL, sorted by URL)

Much more efficient w/o shuffling data
Opportunities for optimizing N-ary operators

- Pushing partition scheme from one input to others: when inputs are partitioned in compatible way this method might be better.
- Heuristic Range partition: Obtaining a overall histogram buckets and generate boundary based on the overall statistics.
- Broadcast optimization: Based common prefix, partition the smaller input and for each partition of large inputs, send all partitions of smaller input to it.
Experimental Results

```
SELECT domain, host, Agg(col1), ..., Agg(coln)
FROM SSTREAM "WebPages.ss"
GROUP BY domain, host
```

<table>
<thead>
<tr>
<th>Domain</th>
<th>Host</th>
<th>Top-level-directory</th>
<th>URL-suffix</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.microsoft</td>
<td>windows</td>
<td>products/</td>
<td>home</td>
<td>...</td>
</tr>
<tr>
<td>com.bing</td>
<td>www</td>
<td>videos/</td>
<td>browse?FORM=Z9LH6</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1: Sample Information for a Web-pages Structured Stream

(a) Latency

(b) Total Work

(c) Data Write and Data Read
The data is ranged-partitioned and sorted by \{domain, host, top-level-directory\}

T_1,T_2,T_3,T_4, come from different period of time and different domain.

```
SELECT domain, host, Agg(col1), ..., Agg(coln)
FROM (  
    SELECT * FROM T1 UNION ALL  
    SELECT * FROM T2 UNION ALL  
    SELECT * FROM T3 UNION ALL  
    SELECT * FROM T4  
)  
GROUP BY domain, host
```
Experimental Results

(b) Total Work

(c) Total Data I/O
Indexed-based Partitioning

- In the situation of terabytes of data, even the local repartition would be quite expensive.
- We could compute a value $p_a$ (index number) utilize a stable sort to virtually “partition” the input data.
Experimental Results

- Original Repartition
- Indexed-based Repartition

![Graph showing the normalized average latency per partition vertex vs. number of partitions. The graph compares Original Repartition and Indexed-based Repartition.]
Critiques and Discussion
The paper did not provide detailed example and description for optimization opportunities for the N-ary operator.

Due to commercial reason, the paper only provides relative measurements for the experiment results.

Network environment for the experiments is not mentioned.
No example and experimental results were given for expensive N-ary operation like join.

All of these advanced partitioning techniques and even the whole optimizer rely heavily on structural properties of the input stream.