Large-Scale Data Analysis: Bridging the Gap

Alekh Jindal, Yagiz Kargin, Sarath Kumar, Vinay Setty
Outline

- Motivation: Parallel DBMS vs Map/Reduce
- Schema & Benchmarks Overview
- Original(Pavlo) Map/Reduce Plans
- Improved(SAVY) Design & Implementation
- Improving Hadoop
  - Indexing
  - Co-Partitioning
- Experiments
- Conclusion
Motivation

- Ever growing data
  - About 20TB per Google crawl!

- Computing Solutions
  - High-end server: 1625.60€/core, 97.66€/GB
  - Share-nothing nodes: 299.50€/core, 166.33€/GB

- Two Paradigms
  - Parallel DBMS
  - Map/Reduce
Parallel DBMS

Parallel DBMS: Advantages

- Can be column based
  - Example: Vertica
- Local joins possible
  - Partition based on join key
- Can work on compressed data
  - reduced data transfer
- Flexible query plans
- Supports Declarative languages like SQL
Parallel DBMS - Shortcomings

● Not free of cost
● Not open source
● Cannot scale to thousands of nodes: why?
  ○ Less fault tolerant
  ○ Assumes homogeneous nodes
● Not so easy to achieve high performance
  ○ Needs highly skilled DBA
  ○ Needs high maintenance
Map/Reduce (Hadoop):

Advantages

- Free of cost
- Open source
- Fault tolerant
- Scales well to thousands of nodes
- Less maintenance
- Flexible query framework
Map/Reduce (Hadoop):

Shortcomings

- Lack of inbuilt Indexing
- Cannot guarantee local joins
- Performance degradation for SQL like queries
  - Multiple MR phases
  - Each MR phase adds extra cost
- No Flexible query plans
- Data transfer not optimized
Benchmarks and Schema
CREATE TABLE Documents (  
    url VARCHAR (100) PRIMARY KEY,  
    contents TEXT  
);  

CREATE TABLE Rankings (  
    pageURL VARCHAR (100) PRIMARY KEY,  
    pageRank INT,  
    avgDuration INT  
);
CREATE TABLE UserVisits (  
sourceIP VARCHAR(16),  
destURL VARCHAR(100),  
visitDate DATE,  
adRevenue FLOAT,  
userAgent VARCHAR(64),  
countryCode VARCHAR(3),  
languageCode VARCHAR(6),  
searchWord VARCHAR(32),  
duration INT  );
Benchmarks 1&2

- **Selection task (Benchmark 1)**
  - `SELECT pageURL, pageRank FROM Rankings WHERE pageRank > X;`

- **Aggregation task (Benchmark 2)**
  - `SELECT sourceIP, SUM(adRevenue) FROM UserVisits GROUP BY sourceIP;`
  - `SELECT SUBSTR(sourceIP, 1, 7), SUM(adRevenue) FROM UserVisits GROUP BY SUBSTR(sourceIP, 1, 7);`
Benchmark 3: Join Task

● SELECT INTO Temp sourceIP, AVG (pageRank) as avgPageRank, SUM (adRevenue) as totalRevenue
FROM Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL AND UV.visitDate BETWEEN Date('2000-01-15') AND Date('2000-01-22') GROUP BY UV.sourceIP;

● SELECT sourceIP, totalRevenue, avgPageRank FROM Temp ORDER BY totalRevenue DESC LIMIT 1;
Original (Pavlo) MR Plans
SELECT pageURL, pageRank FROM Rankings WHERE pageRank > 10;
Benchmark 2: Phase 1

Phase 1

Mapper
Map: split
Map: split
Map: split

Combiner
sum
sum
sum

Reduce:
Reduce: Aggr
Reduce: Aggr
Reduce: Aggr

Result1
Result1
Result1

HDFS
Data
Data
Data

Phase 1

Inter. Resu
Inter. Resu
Inter. Resu

Mapper

Combiner

Reduce

Result1
Benchmark 2: Phase 2

Extra MR job to merge

Phase 2

HDFS

Mapper

Reducer

Result1

Result1

Result1

Identity

Identity

Identity

Reduce

Results
Benchmark 3 – Phase 1

Also classifies two types of records

Also classifies two types of records

HDFS

Mapper

Reduce

User visits

User visits

User visits

Inter. Result

Inter. Result

Inter. Result

Predicates

Predicates

Predicates

Phase 1

Result1

Result1

Result1

<Source IP, URL, PageRank, adRevenue>

<Source IP, URL, PageRank, adRevenue>

<Source IP, URL, PageRank, adRevenue>

Also classifies two types of records
Benchmark 3 – Phase 2

Phase 2

Mappers

Identity

Inter. Resu

Reducers

Avg(PR), Sum (adRevenue)

Avg(PR), Sum (adRevenue)

Avg(PR), Sum (adRevenue)

Result1

Result1

Result1

Result1

Result2

Result2

Result2

Result2

Identity

Identity

Identity

HDFS

<Sourc e IP, URL, PageRank, adRevenue>
Benchmark 3 – Phase 3

HDFS

Mapper

Reducer

Result2

Identity

Inter. Resu

Max(Sum(adRevenue))

Final Result

Source IP, Avg(PR), Sum(adRevenue)

Phase 3

<Maps

Result2

Identity

Inter. Resu

Result2

Identity

Inter. Resu

Result2

Identity

Inter. Resu

<Source IP, Avg(PR), Max(Sum(adRevenue))>

<Source IP, Avg(PR), Max(Sum(adRevenue))>

<Source IP, Avg(PR), Max(Sum(adRevenue))>
Improved (Savy) MR Plans
Binary Data

- Eliminates delimiters
- Avoids splitting
- Makes tuples of fixed length
- Helps in indexing
Benchmark 1

HDFS

Data

Data

Data

Binary data

Phase 1

Mappers

PageRank > 10?

PageRank > 10?

PageRank > 10?

Phase 2

Reducer

Reducer

Reducer

Extra MR job to merge results

Result

Result

Result

Result
Benchmark 2

Phase 1

Mapper

Combiners

Phase 2

Reducer

Extra MR job

to merge

results

Phase 2

Data

HDFS

split

sum

split

sum

split

sum

Binary data

Reducers
Benchmark 3 (Design I) – Phase 1

HDFS

User visits

Rankings

Record Reader

Mappers

Reducers

Easy to classify (just look at record size)

Binary data

<Source IP, URL, PageRank, adRevenue>

Result1

<Source IP, URL, PageRank, adRevenue>

Result1

<Source IP, URL, PageRank, adRevenue>

Result1

Phase 1
Benchmark 3 (Design I) – Phase 2

Mapper
- Identity
- $\text{Avq(PR)}$, $\text{Sum}$

Combiner
- $\text{Avq(PR)}$, $\text{Sum}$

Reducer
- $\text{Max}(\text{Sum}(\text{adRevenue}))$

Final Result

No Phase 3!
Benchmark 3 (Design II) –

Phase 1

**HDFS**

**User visits**

**User visits**

**Only UserVisits**

Very small data (Top R records)
Benchmark 3 (Design I) – Phase 2

HDFS

<Source IP, Sum(adRevenue), <Dest URLs>>

Result1 Rankings

<Source IP, Sum(adRevenue), <Dest URLs>>

Result1 Rankings

<Source IP, Sum(adRevenue), <Dest URLs>>

Result1 Rankings

Max(sum(adRevenue)) & Join

Source IP, Avg(PR), Sum (adRevenue)

Final Result
Improving Hadoop
Improving Hadoop

- Improve Selection (Indexing)
- Improve Join (Co-partitioning)
Indexing

● Data Loading
  ○ index and load data into DFS

● Query Execution
  ○ index look-up and selection

● Implementation on Hadoop
Data Loading

- Partitioning
- Sorting
- Bulk Loading
- HID Splits
Data Loading

Client

Partitioner

Sorting

Bulk load the Index

Construct HID Split

Mapper

Sorting

Bulk load the Index

Construct HID Split

Mapper

Sorting

Bulk load the Index

Construct HID Split

Mapper
Partitioning

Split input data at tuple boundaries
Partitioning

Split input data at tuple boundaries
Partitioning

Split input data at tuple boundaries
Partitioning

Split input data at tuple boundaries
Sorting

Sort each split on the index key
Sorting

Sort each split on the index key
Bulk Loading

Bulk load CSS tree index
HID Split

Construct *Header-Index-Data* Split
HID Split

Construct *Header-Index-Data* Split
HID Split

Construct *Header-Index-Data* Split

- **Header:**
  - Index end offset
  - Data end offset
  - Start index key
  - End index key
HID Split

Construct **Header-Index-Data** Split

Header:  Index end offset
Data end offset
Start index key
End index key
Query Execution

- Partitioning
- Split selection
- Index lookup
- Extractor
Query Execution

Client

Partitioner

Split Selection

Index Lookup

Extractor

Map

Mapper

Mapper

Mapper
Partitioning

Read header to get HID boundaries
Partitioning

Read header to get HID boundaries
Partitioning

Read header to get HID boundaries
Partitioning

Read header to get HID boundaries
Split Selection

Discard splits containing out of range index keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys

Full Contained
Index Lookup

Find data offsets corresponding to LOW and HIGH keys

- Full Contained
- Left Contained
Index Lookup

Find data offsets corresponding to LOW and HIGH keys

- Full Contained
- Left Contained
- Right Contained
Index Lookup

Find data offsets corresponding to LOW and HIGH keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys
Index Lookup

Find data offsets corresponding to LOW and HIGH keys

- **Point Contained**
- **Span**
- **Not Contained**
Extractor

Perform selection on data
Extractor

Pass sub-split to Record Reader for processing
Implementation on Hadoop

Loading
- CSS Tree Index
- Indirect index
- Four key types supported - Int, Float, Date, String
- Index stored as byte array
- Reducer to reduce number of files
- Integral number of HID splits per reducer output

Querying
- Discover HID split boundaries from respective headers
- Read only the selected data from HDFS
Co-Partitioning

- Data loading
- Query execution
Data Loading

Client

Map

Map

Map

Reduce

Reduce

Group on Join key

Add header to co-group
Data Loading

Relation 1

Relation 2
Data Loading

Relation 1

Relation 2

Group by Join key (Map)

Group 1
Group 1

Group 2
Group 2

Group 3
Group 3
Data Loading

Relation 1

Relation 2

Group by Join key (Map)

Group 1 Group 1

Group 2 Group 2

Group 3 Group 3

Add Header (Reduce)

H Group 1 Group 1

H Group 2 Group 2

H Group 3 Group 3
Data Loading

Relation 1

Relation 2

Group by Join key (Map)

Group 1 Group 1

Group 2 Group 2

Group 3 Group 3

Add Header (Reduce)

Group 1 Group 1

Group 2 Group 2

Group 3 Group 3

Combine into single file (Reduce)

Group 1 Group 1

Group 2 Group 2

Group 3 Group 3
Query Execution
Query Execution

Mapper

Mapper

Mapper

Local Join

Local Join

Local Join
Indexing on top of Co-partitioning
Indexing on top of Co-partitioning
Indexing on top of Co-partitioning
Indexing on top of Co-partitioning
Indexing on top of Co-partitioning

Mapper

Re-arrange

Mapper

Re-arrange
Indexing on top of Co-partitioning

Re-arrange

Index

Mapper

Mapper
Indexing on top of Co-partitioning
Experiments
Experimental Setup

- Hadoop 0.19.1
- 5 nodes
- Speed?
- RAM?
- Gigabit Ethernet
- Data size
  - User Visits: 20GB
  - Rankings: 32MB
Results

Data Size

- Text: 32M
- Binary: 64M

- Rankings
- UserVisits

- Data Size: 19G & 35G
Results

Benchmark 1

<table>
<thead>
<tr>
<th></th>
<th>Pavlo</th>
<th>Savy</th>
<th>Savy-binary</th>
<th>Savy-Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time(s)</td>
<td>32</td>
<td>24</td>
<td>22</td>
<td>30</td>
</tr>
</tbody>
</table>
Results

Benchmark2

<table>
<thead>
<tr>
<th></th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pavlo</td>
<td>200.01</td>
</tr>
<tr>
<td>Savy</td>
<td>168.722</td>
</tr>
<tr>
<td>Savy-binary</td>
<td>212.048</td>
</tr>
</tbody>
</table>

Legend:
- Pavlo
- Savy
- Savy-binary
Results

![Benchmark 3 Results Chart]

<table>
<thead>
<tr>
<th>Time(s)</th>
<th>Pavlo</th>
<th>Savy</th>
<th>Savy-binary</th>
<th>Savy-Index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>152.019</td>
<td>127.354</td>
<td>175.28</td>
<td>103.568</td>
</tr>
</tbody>
</table>
Results

![Benchmark 1 chart showing results for different methods.](chart.png)
Roadblocks Faced

- **Data generation:**
  - 20GB UserVisits, 338MB Rankings in HDFS
  - Took 16 hours for generation
  - Too many OS/library dependencies
  - Poor documentation

- **Number of nodes:**
  - Allocated 6 nodes
  - Effective (up-and-running) 4 nodes
  - Map/Reduce parallelism not exploited
  - Per-split indexing ideally suited for highly parallel execution
Roadblocks Faced

- Data normalization
  - Schema uses VARCHAR data types
  - Input data normalized to fixed tuple-sized binaries
  - Byte oriented processing speedup negated by increased input size
  - However, facilitates indexing and co-partitioning

- Low selectivity
  - Selection task has selectivity close to 1
  - Indexing benefits are sabotaged

- Incorrect base result
  - Reported join task result was not correct
Roadblocks Faced

- Implementation deviation from the paper
  - Composite key is not really used in join task
Discussion: Loopholes

- Benchmarks are well suited (biased) for databases
- Huge difference in data loading time
- Queries make heavy use of indexing, sorting data
- Query optimization not done for Map/Reduce
- Fault tolerance not compared
Discussion: We can do better!

- Map/Reduce plans can be optimized
- Normalized binary input data can help
- Indexing feasible and performs good
- Co-partitioning feasible and looks promising
References
