Novel Selectivity Estimation Strategy for Modern DBMS

Jun Hyung Shin[1], Florin Rusu[1], Alex Şuhan[2]

University of California, Merced[1], MapD Technologies, Inc[2]

jshin33, frusu@ucmerced.edu; alex@mapd.com

Selectivity Estimation Problem

$$\sigma_{A=x \text{ AND } B>y1 \text{ AND } B<y2 \text{ AND } (C=z1 \text{ OR } C=z2)}(\boldsymbol{R})$$

Selectivity estimation requires database statistics, such as:

$\boldsymbol{T(R)}$ = # tuples of relation $\boldsymbol{R}$

$\boldsymbol{V(R,A)}$ = # distinct values relation $\boldsymbol{R}$ has in attribute $\boldsymbol{A}$

Estimated Selectivity:

$$T(\sigma_{...}(R)) = T(R) \times \frac{1}{V(R,A)} \times \frac{1}{3} \times \frac{1}{3} \times \left(\frac{1}{V(R,C)} + \frac{1}{V(R,C)}\right)$$

Assumption: each predicate is independent + uniform data

- Not realistic in real world
- Estimation can be **wrong** by orders of magnitude
- Results in a **bad query execution plan**

Estimation is **not accurate with complex predicates**

Selectivity can be better estimated by other methods:

- Histograms & Sketches
  - better estimation for a single attribute
  - still bad on multiple attributes
- Samples
  - number of attributes does not matter
  - bad on skewed & sparse data

Modern Database Systems

I/O bottleneck when read from disk was critical in the past

Evolution of modern database systems:

- **In-memory**: much less I/O access
- **GPU-accelerated**: massively parallel architecture
  better at running aggregate queries

Main Idea

Focus on accuracy:

$\boldsymbol{T(\sigma_{...}(R))} =$
```
SELECT COUNT(*)
FROM R
WHERE A = x AND B < y1 AND B > y2
AND (C = z1 OR C = z2);
```

Running this query will return **exact selectivity** and would not take much time in modern database systems

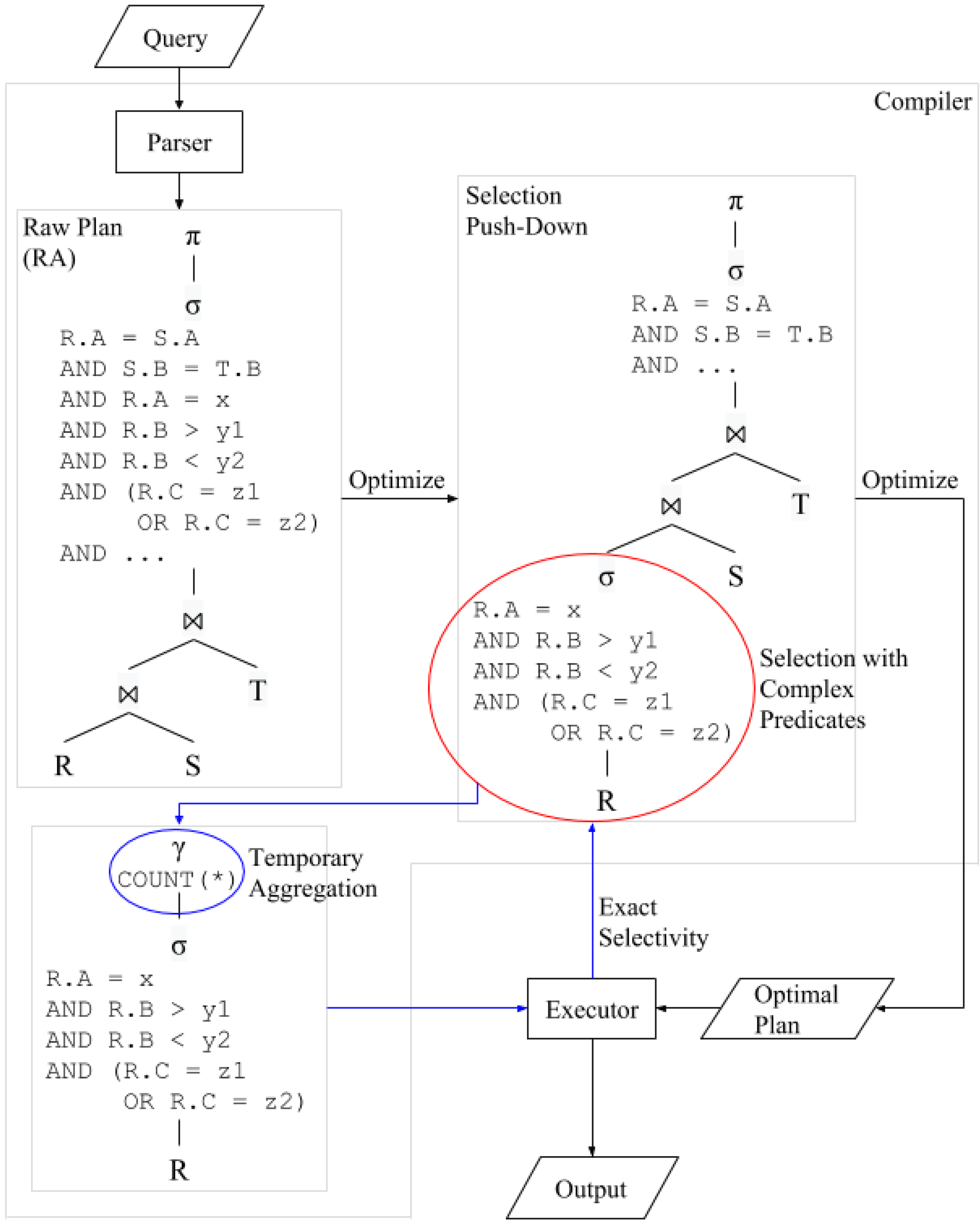
Effective for complex predicates

Does not require any database synopses and statistics

Consider the trade-off between accuracy and time

- Cost of time is still relatively expensive
- Hybrid approach for better optimization

Generate and Execute Extra Plans



Selection Push-Down Decision

MapD: In-memory, GPU-accelerated, column-oriented database system

MapD *does not* perform selection push-down

- high selectivity → **huge materialization** → system slows down
- requires accurate selectivity estimation to prevent such selection

Compute **exact selectivity** by running an extra plan for each selection

- **avoid** push-down on **small relations** < `PUSH_DOWN_MIN_TABLE_SIZE`
- push-down relevant projection to read only necessary columns
- push-down *only if* selectivity < `PUSH_DOWN_MAX_SELECTIVITY`
- execute selection immediately and store the output as temporary table so that the system can reuse it while executing original query

Experiments

TPC-H Benchmark dataset on MapD

2 Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz, 1 Tesla K80 GPU, 8 DDR4 memory 32GB @ 2400 MHZ

Overhead:

by scale-factor and relations (ms)

SF	50				10				1			
Tables	O	PS	P	S	O	PS	P	S	O	PS	P	S
GPU	21	18	21	19	20	23	23	28	21	20	22	22
CPU	70	95	66	57	49	58	25	21	21	23	24	21

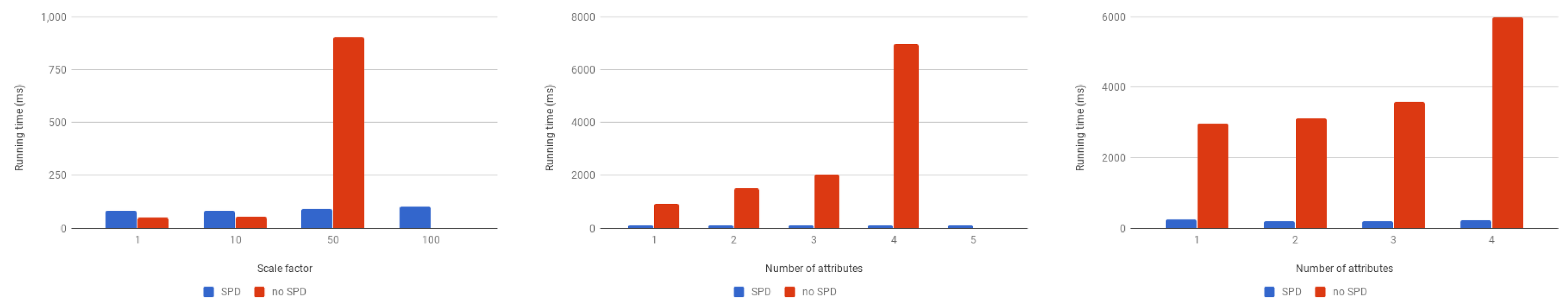
by selectivity (ms)

Selectivity	1	75M
GPU	21	22
CPU	70	66

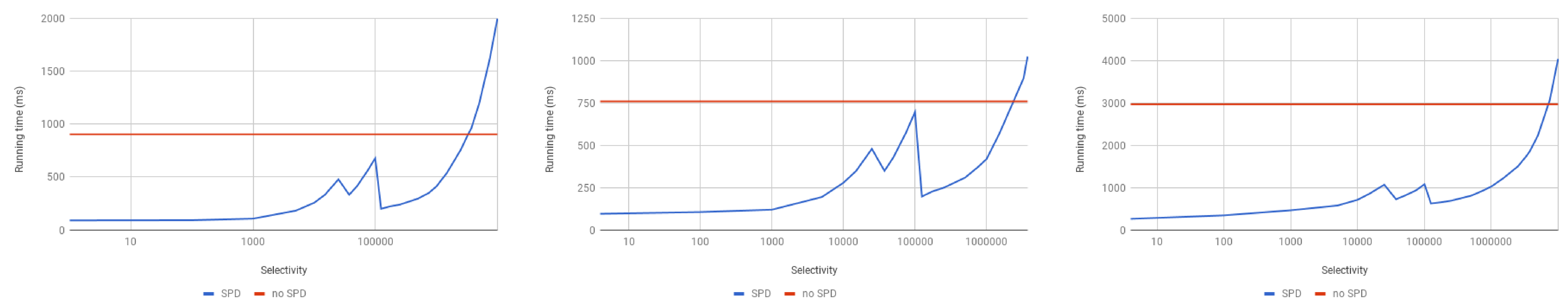
by # attributes (ms)

# attributes	1	2	3	4
GPU	16	23	23	18
CPU	70	91	273	339

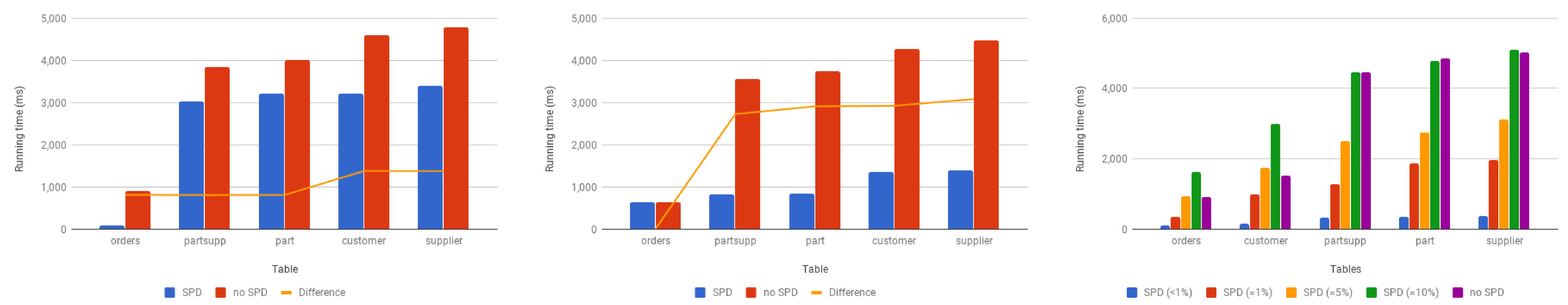
Selection Push-Down Decision:



by scale factor   by # attributes



by selectivity



by # tables   complex case