Toward timely, predictable and cost-effective data analytics

Renata Borovica-Gajić
DIAS, EPFL
Big data proliferation

The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020

“Big data is when the current technology does not enable users to obtain **timely, cost-effective, and quality** answers to **data-driven questions**. “ [Steve Todd, Berkeley]

Technology follows Moore’s Law ™


Ț “Trends in big data analytics”, 2014, Kambatla et al
What business analysts want

Timely, predictable, cost-effective queries

- 80% reuse within 3 hours
- Minimal data-to-insight time
- Predictable response time
- Low infrastructure cost

[WinterCorp, 2013]
Thesis statement

As traditional DBMS rely on predefined assumptions about workload, data and storage, changes cause loss of performance and unpredictability.

Insight

Query execution must adapt at three levels (to workload, data and hardware) to stabilize and optimize performance and cost.
Outline

• Minimize data-to-insight time
  – Workload-driven adaptation [SIGMOD’12, VLDB’12, CACM’15]

• Improve predictability of response time
  – Data-driven adaptation [DBTest’12, ICDE’15]

• Reduce analytics cost
  – Cold storage & hardware-driven adaptation [VLDB’16]
Outline

• Minimize data-to-insight time
  – Workload-driven adaptation

• Improve predictability of response time
  – Data-driven adaptation

• Reduce analytics cost
  – Cold storage & hardware-driven adaptation
Current technology ≠ efficient exploration

Data-to-insight time

Traditional query stack

- Loading
- Querying
- Insight

Raw data querying stack

- Processing (Q1)
- Convert
- Tokenize
- Parse
- I/O

Time to first insight too long
Does not scale with data growth

Overheads too high

Overheads too high
Optimize raw data querying stack

Raw data querying stack

- Processing (Q1)
- Convert
- Tokenize
- Parse
- I/O

NoDB: Workload-driven data loading & tuning

Let users show by asking queries

Not everything needed for Q1
PostgresRaw: NoDB from idea to practice

1. Positional indexing

Pointers to attributes

Pointers to end of tuples

2. Cache

NationKey

17
5
...

1|Supplier#01|17|335-1736|5755.94|each slyly...
2|Supplier#02|5|861-2259|4032.68|slyly bold...
3|Supplier#03|1|516-1199|4192.40|blithely...
...1|Supplier#04|15|787-7479|4641.08|riously eve...
...1|Supplier#05|11|21-151-690-3663|-283.84|
...Slyly... 6|Supplier#06|14|24-696-997-4969|1365.79|final...

3. Statistics

Frequency

0 2 5 10

1 3 5 7 9 11 13 15

# Buckets

Adjust to queries = progressively cheaper access
**PostgresRaw in action**

**Setting:** 7.5M tuples, 150 attributes, 11GB file  
**Queries:** 10 arbitrary attributes per query, vary selectivity

**Data-to-insight time halved with PostgresRaw**
**Per query performance comparable to traditional DBMS**
Summary of PostgresRaw

- Query processing engine over raw data files
- Uses user queries for partial data loading and tuning
- Comparable performance to traditional DBMS

IMPACT

- Enables timely data exploration with 0 initialization
- Decouples user interest from data growth
Outline

• Minimize data-to-insight time
  – Workload-driven adaptation

• Improve predictability of response time
  – Data-driven adaptation

• Reduce analytics cost
  – Cold storage & hardware-driven adaptation
Index: with or without?

**Setting:** TPC-H, SF10, DBMS-X, Tuning tool 5GB space for indexes

![Bar chart showing normalized execution time for TPC-H queries with and without indexes.](chart.png)

**Performance hurt after tuning**
Access path selection problem

Re-optimization: risky

Access path selection problem

Execution time

Performance cliff

RISK

Statistics: unreliable advisor
Re-optimization: risky

Index Scan
Full Scan

[MI’98, POP’04, RIO’05, BOU’14]
Quest for predictable execution

Removing variability due to (sub-optimal) choices
Smooth Scan

**Morph** between Index and Sequential Scan based on *observed result* distribution
Morphing mechanism

Modes:

1. **Index Access**: Traditional index access
2. **Entire Page Probe**: Index access probes entire page
3. **Gradual Flattening Access**: Probe adjacent region(s)
Morphing policy

- Selectivity Increase -> Mode Increase
- Selectivity Decrease -> Mode Decrease

\[ SEL_{region} \geq SEL_{global} \]
\[ SEL_{region} < SEL_{global} \]

Region snooping = Data-driven adaptation
Smooth Scan in action

**Setting:** Micro-benchmark, 25GB table, Order by, Selectivity 0-100%

Near-optimal over entire selectivity range
Summary of Smooth Scan

- Statistics-oblivious access path
- Uses region snooping to morph between alternatives
- Near-optimal performance for all selectivities

IMPACT

- Removes access path selection decision
- Improves predictability by reducing variability in query execution
Outline

• Minimize data-to-insight time
  – Workload-driven adaptation

• Improve predictability of response time
  – Data-driven adaptation

• Reduce analytics cost
  – Cold storage & hardware-driven adaptation
Proliferation of cold data

“80% enterprise data is **cold** with 60% CAGR” [Horison, 2015]
“cold data: incredibly valuable for analysis” [Intel, 2013]

Cold Storage Devices (CSD) to the rescue

PB-size storage at cost ~ tape and latency ~ disks
CSD in the storage tiering hierarchy

Tiers

- DRAM
- SSD
- 15k RPM HDD
- 7200 RPM HDD
- Tape

Performance

Capacity

Archival

Data Access Latency:

ns µs ms sec min hour

Performance:

- $$$

Capacity:

- $$

Archival:

- $

15k RPM HDD

7200 RPM HDD

Tape

19.3
CSD in the storage tiering hierarchy

Can we shrink tiers to reduce cost?
CSD in the storage tiering hierarchy

Can we shrink tiers to reduce cost?
CSD in the storage tiering hierarchy

CSD offer significant cost savings (40%)
But ... can we run queries over CSD?
Query execution over CSD

Setting: virtualized enterprise datacenter, clients: PostgreSQL, TPCH 50, Q12, CSD: shared, layout: one client per group

Lost opportunity: CSD relegated to archival storage
Skipper to the rescue

Virtualized enterprise data center

I/O Scheduler

Network

VM1

VM2

DB1

DB2

DB3

Object-group map.

Multi-way joins:
Opportunistic execution triggered upon data arrival

1. Novel ranking algorithm:
   Balances access efficiency across groups and fairness across clients

2. Progress driven caching:
   Favors caching of objects to maximize query progress

3. Cache Management

   MJoin

   Hash

   Hash

   Hash

   Scan A

   Scan B

   Scan C

   A1

   A2

   B1

   C1
**Skipper in action**

**Setting:** multitenant enterprise datacenter, clients: TPCH 50, Q12, CSD: shared, layout: one client per group

Approximates HDD-based capacity tier by 20% avg.
Summary of Skipper

• Efficient query execution over CSD with:
  1. Rank-based I/O scheduling
  2. Out-of-order execution based on multi-way joins
  3. Progress based caching policy

• Approximates performance of HDD-based storage tier

IMPACT

• Cold storage can reduce TCO by shrinking storage hierarchy
• Skipper enables data analytics-over-CSD-as-a-service
Thesis contributions

• Minimize data-to-insight time
  – Workload-driven adaptation
  – Skip loading, tune as a byproduct of query execution

• Improve predictability of response time
  – Data-driven adaptation
  – Remove access decisions a priori, transform gradually

• Reduce analytics cost
  – Cold storage & hardware-driven adaptation
  – From plan pull-based to hardware push-based execution

• Uncertainty cured with adaptivity

Thank you!