Cheetah: Accelerating database queries with switch pruning

Muhammad Tirmazi, Ran Ben Basat, Jiaqi Gao, Minlan Yu
(Harvard University)

Published in ACM SIGMOD 2020
Databases

Over 8 billion daily queries on Alibaba cloud
Spark control flow

Spark Master

Query Planner

Spark Workers

Spark Workers

Spark Workers

A, B

A, B, C, E

A, C, D, E

A, B, C, D, E

A, B, A, A

A, B

C, E, A, B, C, C

A, B, C, E

D, A, E, A, D, A, C

A, C, D, E

A, B, C, D, E

DISTINCT
The problem

DB workload growth - fast
Generic CPUs - not fast enough

Cheetah's approach:
Offload to **programmable switches**
Why programmable switches?

Process TBs of data - packets in microseconds
Why programmable switches?

Process TBs of data - packets in microseconds

Already in the network.

Process cross-partition data.
PISA programmable switch

Stateful memory

Match-Action pipeline

Constraint 1: Limited memory

Constraint 2: Limited stages

One way
Distinct query
Distinct query

Switch

A

B

C

A
Append new entry

Switch

A
B
C
D

D
Issue: limited stages.
The pruning abstraction

Dataset: A, B, A, A, C, D, D, E, A, B, C, C, D, A, E, A, D, A, C
**The pruning abstraction**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>A, B, A, A, C, D, D, E, A, B, C, C, D, A, E, A, D, A, C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpruned dataset</td>
<td>A, B, A, C, D, E, B, A, E, A, A, C</td>
</tr>
<tr>
<td>dataset at switch</td>
<td></td>
</tr>
</tbody>
</table>
The **pruning** abstraction

Dataset

A, B, A, A, C, D, D, E, A, B, C, C, D, A, E, A, D, A, C

Unpruned dataset at switch

A, B, A, C, D, E, B, A, E, A, A, C

Result

A, B, C, D, E

Query on **unpruned** dataset = Query on original dataset
Integrating pruning with Spark
Solution 1: Bloom filter

New key hashed to bits in switch memory

Same key arrives again - pruned by switch

Issue: false positives break pruning guarantee!
Solution 2: cache

Switch

ABCDE
False negatives: **not** an issue!

Switch

E B C D E
Issue: partitioned memory

Switch
Rolling replacement insertion

Switch

EOBR

E D C B
Rolling replacement insertion

Switch

\[
\begin{array}{cccc}
D & Q & B & A \\
C & D & D & C \\
\end{array}
\]
Rolling replacement insertion

Switch

DCBA

Least Recent
Issue: pruning rate

Switch 1 value per stage

Less than 50% of duplicates pruned for all tested workloads
**Solution: multi-row cache**

Stateful memory

<table>
<thead>
<tr>
<th>Cache 0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cache 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cache 3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Over 99% of duplicates pruned with only 1024 rows
### Top-N query

Select largest 10 entries in a column.

**Stateful memory**

<table>
<thead>
<tr>
<th>Minima 0</th>
<th>Minima 1</th>
<th>Minima 2</th>
<th>Minima 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Minimum of first 10 entries**

<table>
<thead>
<tr>
<th>Minima 0</th>
<th>Minima 1</th>
<th>Minima 2</th>
<th>Minima 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Multiple pruning **thresholds**

Select largest 10 entries in a column.

Stateful memory

| Minima 0 | 0 | 0 | 0 |
| Minima 1 | 1 | 1 | 1 |
| Minima 2 | 2 | 2 | 2 |
| Minima 3 | 3 | 3 | 3 |

Saw 10 entries larger than 8

**hash**
Randomly assign row

Select largest 10 entries in a column.

Stateful memory

| Minima 0 | 0 | 0 | 0 |
| Minima 1 | 1 | 4 | 1 | 16 |
| Minima 2 | 2 | 2 | 2 |
| Minima 3 | 3 | 3 | 3 |

Saw 10 entries larger than 8
Group-By query

Select MAX(v) from table GROUP BY k

(A, 5) → (A, 5) → (A, 5) → (A, 5)
Different hash for each column

Select MAX(v) from table GROUP BY k
Pruning policy

Select \( \text{MAX}(v) \) from table \( \text{GROUP BY } k \)
Updating pruning points

Select \( \text{MAX}(v) \) from table \( \text{GROUP BY } k \)
Join query

Select * from A JOIN B ON A.x = B.x
Pruning policy

Select * from A JOIN B ON A.x = B.x

Bloom filter

Entry from B

Not in Bloom filter? Prune.
Pruning policy

Select * from A JOIN B ON A.x = B.x

Bloom filter

Entry from B

In Bloom filter? Don't prune.
## Queries supported

<table>
<thead>
<tr>
<th>Query</th>
<th>Distinct</th>
<th>Join</th>
<th>Group-By</th>
<th>Top-N</th>
<th>Having</th>
<th>Filtering</th>
<th>Skyline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Partitioning</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Caches</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bloom filters</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sketches</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thresholds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial query offloading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Projection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

*Store helpful pruning points*
Implementation

Cheetah Master & Cheetah Worker
- 3600 lines of DPDK-optimized C code
- Transparent to Spark user.

Pruning
- 1800 lines of P4
- O(10s) control plane rules / query

github.com/harvard-cns/cheetah-release
Experimental setup

- Programmable switch: Intel's Barefoot Tofino

- Default Spark 2x deployment - 5 workers

- Big Data Uservisits: 6.4 Million Entries ~ 1.3 M per partition

- 10 Gbps network bandwidth. 2 cores / 4 GB per worker.
Pruning optimizes **compute bound** queries.

40% to 75% **faster** completion times
The Network-Compute Tradeoff
Also in the paper

Pruning algorithms for **Having**, Skyline, and Filtering along with workload-independent pruning rate guarantees.

**Reliability** protocol that supports pruning.

Support for **compound queries** and **multiple switches**.
The **network** should play an **active role** in **query processing**.

Switches have **constraints**, but can optimize queries with the right abstraction: **pruning**.

[Cheetah](https://github.com/harvard-cns/cheetah-release) (CC BY 2.0) by wwarby