Cheetah: Accelerating database queries with switch pruning

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

ACM SIGMOD 2020
Databases

Over 8 billion daily queries on Alibaba cloud

"BigQuery" (CC-BY 4.0) by Google LLC
Spark control flow

Spark Master → Query Planner

Query Planner → Spark Workers

Spark Workers → Spark Workers

Spark Workers → Spark Workers

Spark Workers → Spark Workers

Spark Master → Spark Workers

Spark Master → Spark Workers

DISTINCT

A, B, A, A

A, B

A, B, C, E

A, C, D, E

A, B, C, D, E

A, B

A, B, C, E

A, C, D, E

A, B, C, D, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E

A, B, C, E
The problem

DB workload growth - fast
Generic CPUs - not fast enough

Cheetah's approach:
Offload to programmable switches

"Cheetah" (CC BY 2.0) by wwarby
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

ABCD

D
Issue: limited stages.

Switch

A B C D
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 at switch</td>
<td>A, B, A, C, D, E, B, A, E, A, A, C</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

EBCD

E
Issue: partitioned memory
Rolling replacement insertion

Switch

EOBR

EDCB
Rolling replacement insertion

Switch

C D

A B D
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**
### 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
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

"Cheetah" (CC BY 2.0) by wwarby
The Network-Compute Tradeoff
Also in the paper

Pruning algorithms for Join, Group-By, Having, Skyline, Top-K, and Filtering along with workload-independent pruning rate guarantees.

Reliability protocol that supports pruning.

Support for compound queries and multiple switches.
Takeaway

Questions?

The **network** should play an **active role** in **query processing**.

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

```
github.com/harvard-cns/cheetah-release
```

"Cheetah" (CC BY 2.0) by wwarby