CS 764: Topics in Database Management Systems
Lecture 26: Pushdown DBMS

Xiangyao Yu
12/5/2022
Announcements

DAWN workshop
- Reserve a presentation slot using the following google sheet
  https://docs.google.com/spreadsheets/d/1Re1M9FmJwl_YkidhNgeV0iKn-clssFrK_J1PMidaAuw/edit?usp=sharing
- 8-min per group (presentation + QA)

Project report (DDL: Dec. 19)
- **Submit to the hotcrp website** (like the proposal)

Submit course evaluation on aefis.wisc.edu
PushdownDB: Accelerating a DBMS Using S3 Computation

Xinyao Yu*, Min You**, Matthew Wooldridge1, Avinash Gianchandani2, Marco Scaillet1, Ashraf AboEl-Ata2, Michael Stonebraker1

1University of Pennsylvania, 2University of California, Berkeley

*Symantec, +Oracle, †UC Berkeley, ‡University of California, Berkeley

January 25, 2021

ICDE 2020

FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS

Yiwei Yang†, Matt You**, Matthew Wooldridge1, Yubin Liu†, Xinyao Yu*, Murat Senel2, Ashraf AboEl-Ata2, Michael Stonebraker1

†University of Wisconsin-Madison, ‡University of California, Berkeley

*Symantec, +Oracle, †UC Berkeley, ‡University of California, Berkeley

January 25, 2021

VLDB 2021

ABSTRACT

Commercial databases adopt a complex-decomposition architecture that separates the management of computation and storage. A flexible approach to such a functional separation of S3/S3 Select allows only certain query operators to be pushed into the storage layer, namely selections, projections, and simple aggregations. Other operators require new implementations to take advantage of S3 Select. Moreover, S3 Select’s pricing can be more expensive than computing on normal EC2 nodes.

In this paper, we are going to understand the performance of computation pushdown when running queries in a cloud storage with disaggregated storage. Specifically, we consider blue (with and without indexing), join, group by, and top-k as calculations. We implement these operators to take advantage of computation pushdown through S3 Select and study their costs and performance. We show that this approach improves the performance and cost reduction through the relatively high cost of S3 S3 Select. In addition, we analyze queries from the TPC-H benchmark and show similar benefits of computation and cost. We point out the limitations of the current S3 Select storage and provide several suggestions based on the trends we learned from this project. In the best-of-the-knowledge

ICDE 2020

ICDE 2020

VLDB 2021

ICDE 2020

VLDB 2021

ICDE 2020

VLDB 2021
Storage-Disaggregation Architecture

Features of disaggregation architecture
- Computation and storage layers are disaggregated
- Limited computation can happen in the storage layer
Storage-Disaggregation Architecture

Features of disaggregation architecture

- Computation and storage layers are disaggregated
- Limited computation can happen in the storage layer

Advantages

- Lower management cost
- Independent scaling of computation and storage

Disadvantages

- Network becomes a bottleneck
How to Mitigate the Network Bottleneck?

Solution 1: Move data to computation
- Cache storage data in the computation layer
- Example: Snowflake

Solution 2: Move computation to data
- Pushdown computation to the storage layer
- Example: PushdownDB
Key questions to address in this project:

- How to implement relational operators to leverage existing cloud services?
- What are the performance and cost tradeoffs?
PushdownDB – Building Blocks

PushdownDB implementation
- Single-node, multi-process Python-based database
- Ubuntu 16.04.5 LTS, Python version 2.7.12.

Source code: https://github.com/yxymit/s3filter.git
Simple Cloud Storage (S3)

Virtually infinite storage capacity with relatively low cost

Partition input relations into multiple shards, each shard is stored as a separate object in S3

S3 vs. elastic block store (EBS) vs. local store
  - Virtually infinite capacity, shared across all nodes, lower cost, durable
Supports limited SQL queries on **CSV** and **Parquet** data format

- S3 Select recognizes database schema for both data formats
- **Simple queries with predicates and aggregation** (no join, no group-by, no sort, etc.)
PushdownDB – Supported Operators

S3 Select supports
- Filter
- Project
- Aggregate without group-by

PushdownDB supports
- Filter
- Project
- Top-K
- Join
- Group-by
Server-side filtering

- Compute server loads entire table from S3 and filters locally

Example query:
```
SELECT col1, col2
FROM R
WHERE col1 < 10
```
Filter

Server-side filtering
– Compute server loads entire table from S3 and filters locally

S3-side filtering
– Push down predicate evaluation using S3 Select

Example query:
SELECT col1, col2
FROM R
WHERE col1 < 10
Baseline Join

- Server loads both tables from S3 and joins locally

```sql
SELECT SUM(O_TOTALPRICE)
FROM CUSTOMER, ORDER
WHERE
  O_CUSTKEY = C_CUSTKEY
  AND C_ACCTBAL <= upper_c_acctbal
  AND O_ORDERDATE < upper_o_orderdate
```
Join

Baseline Join
– Server loads both tables from S3 and joins locally

Filtered Join
– Server pushes filtering predicates to S3 to load both tables

```
SELECT SUM(O_TOTALPRICE)
FROM CUSTOMER, ORDER
WHERE
  O_CUSTKEY = C_CUSTKEY
  AND C_ACCTBAL <= upper_c_acctbal
  AND O_ORDERDATE < upper_o_orderdate
```
Bloom Join

– Step 1: Server loads the smaller table, builds a bloom filter using join key
– Step 2: Server sends the filter via S3 Select to load the bigger table
– Bloom filter is pushed down as a predicate

```
SELECT ... 
FROM S3Object 
WHERE SUBSTRING('1000011...111101101', 
  ((69 * CAST(attr as INT) + 92) % 97) % 68 + 1, 1 ) = '1' 
```

```
SELECT SUM(O_TOTALPRICE) 
FROM CUSTOMER, ORDER 
WHERE 
  O_CUSTKEY = C_CUSTKEY 
  AND C_ACCTBAL <= upper_c_acctbal 
  AND O_ORDERDATE < upper_o_orderdate 
```
Evaluation – Join

```
SELECT SUM(O_TOTALPRICE)
FROM CUSTOMER, ORDER
WHERE O_CUSTKEY = C_CUSTKEY
AND C_ACCTBAL <= upper_c_acctbal
AND O_ORDERDATE < upper_o_orderdate
```
Overall, PushdownDB reduces runtime by $6.7 \times$ and reduces cost by 30%
PushdownDB: Accelerating a DBMS Using S3 Computation

Xiaoyi Yu1, Matt You1, Matthew Woeltz1, Abhimanun Ghanshyam1, Mario Serfati1, Adarsh Abhishekar1, Mihalis Stoubouler2

1Microsoft Research; 2University of Massachusetts Amherst
Email: yxyu@microsoft.com, myyou@microsoft.com, mwoeltz@microsoft.com, aghanshyam@microsoft.com, mserfati@umass.edu, mstoubouler@umass.edu

ABSTRACT

PushdownDB adopts a compute-data separation architecture that promotes the management of computation and storage. A dataflow graph is used as an abstraction for the network connecting the computation and storage layers. Two solutions have been explored to reduce the feedback path latency, using S3 for computing and caching pushdown. Both techniques can significantly reduce network bandwidth, improve throughput, and reduce I/O costs.

1 INTRODUCTION

PushdownDB (Sanka1) is a research project to build a system for accelerating data processing applications with S3. It departs from previous efforts in implementing accelerators on S3 (e.g., EuroS3) by using a compute-data separation architecture that promotes the management of computation and storage. The architecture is based on a dataflow graph (DFG) that describes the processing logic, and a data-driven execution model that promotes the separation of computation and storage. The key idea is to use S3 as a storage layer and perform computation on S3 in a compute-data separation architecture.

ICDE 2020

FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS

Yifei You1, Matt You1, Matthew Woeltz1, Yishuo Liu1, Xiaoyi Yu1, Mihalis Stoubouler1, Mario Serfati1, Adarsh Abhishekar1, Michael Steinberger2

1Microsoft Research; 2University of Massachusetts Amherst
Email: yxyu@microsoft.com, myyou@microsoft.com, mwoeltz@microsoft.com, aghanshyam@microsoft.com, mserfati@umass.edu, mstoubouler@umass.edu

ABSTRACT

FlexPushdownDB adopts a compute-data separation architecture that promotes the management of computation and storage. A dataflow graph is used as an abstraction for the network connecting the computation and storage layers. Two solutions have been explored to reduce the feedback path latency, using S3 for computing and caching pushdown. Both techniques can significantly reduce network bandwidth, improve throughput, and reduce I/O costs.

1 INTRODUCTION

FlexPushdownDB (Sanka2) is a research project to build a system for accelerating data processing applications with S3. It departs from previous efforts in implementing accelerators on S3 (e.g., EuroS3) by using a compute-data separation architecture that promotes the management of computation and storage. The architecture is based on a dataflow graph (DFG) that describes the processing logic, and a data-driven execution model that promotes the separation of computation and storage. The key idea is to use S3 as a storage layer and perform computation on S3 in a compute-data separation architecture.

VLDB 2021
Mitigate Network Bottleneck

**Baseline**: always load data from cloud storage (e.g., S3)
- Examples: default presto, hive, SparkSQL, etc.
Mitigate Network Bottleneck

**Baseline**: always load data from cloud storage (e.g., S3)

**Caching**: cache hot table data in the compute node
- Examples: Snowflake, redshift spectrum (static), Alluxio, etc.
**Baseline**: always load data from cloud storage (e.g., S3)

**Caching**: cache hot table data in the compute node

**Pushdown**: push down selection, projection, aggregation to storage
- Examples: Redshift spectrum, Aqua, PushdownDB, etc.
Caching vs. Pushdown

**Caching** performance increases with a bigger cache

**Pushdown** performance is independent of cache size
Caching vs. Pushdown

Caching performance increases with a bigger cache.

Pushdown performance is independent of cache size.

A hybrid design may achieve the best of both worlds.
Mitigate Network Bottleneck

**Baseline (Pullup):** always load data from cloud storage (e.g., S3)

**Caching:** cache hot table data in the compute node

**Pushdown:** push down selection, projection, aggregation to storage

**Hybrid:** hybrid caching and pushdown at fine granularity
Design choices
  – Cache table data rather than query results for simplicity

Source code: https://github.com/cloud-olap/FlexPushdownDB
FlexPushdownDB (FPDB) Overview

Design choices
- Cache table data rather than query results for simplicity
- Segment as the caching granularity

Source code: https://github.com/cloud-olap/FlexPushdownDB
Design choices

- Cache table data rather than query results for simplicity
- Segment as the caching granularity

Source code: https://github.com/cloud-olap/FlexPushdownDB
**FlexPushdownDB (FPDB) Overview**

### Main modules

- **Hybrid executor**
  - Query plan
- **Cache manager**
  - Admission
  - Eviction
- **Local Cache**
- **Caching request**
FlexPushdownDB (FPDB)

Separable operators
- Can execute separately using cached segments and cloud storage
- Example: projection, selection, aggregation, hash join (partially)
FlexPushdownDB (FPDB)

Separable operators
– Can execute separately using cached segments and cloud storage
– Example: projection, selection, aggregation, hash join (partially)

Query execution
– Heuristic: exploit caching when possible, otherwise pushdown as much as possible
Separable Query Plan — Example

SELECT R.B, sum(S.D)
FROM R, S
WHERE R.A = S.C AND R.B > 10 AND S.D > 20
GROUP BY R.B
Cache Manager

Traditional caching assumption: *Equal-size cache misses incur the same cost*
Cache Manager

Traditional caching assumption: **Equal-size cache misses incur the same cost**

In FPDB, misses that cannot exploit pushdown have higher cost, and should be considered for cached with higher priority
Cache Manager

Traditional caching assumption: Equal-size cache misses incur the same cost

In FPDB, misses that cannot exploit pushdown have higher cost, and should be considered for cached with higher priority

**Weighted-LFU** cache replacement policy
- Increment the frequency counter with the estimate miss cost
- Estimated miss cost = network cost + scan cost + compute cost
Performance Evaluation

Conclusion: FPDB outperforms baselines by 2.2x
Evaluation – Weighted-LFU

Weighted-LFU outperforms the baseline LFU by 37%
Table 2: Network Usage (GB) of different architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Pullup</th>
<th>PD-only</th>
<th>CA-only</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>460.9</td>
<td>37.1</td>
<td>112.6</td>
<td>7.9</td>
</tr>
</tbody>
</table>
Evaluation – Resource Usage

Table 2: Network Usage (GB) of different architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Pullup</th>
<th>PD-only</th>
<th>CA-only</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>460.9</td>
<td>37.1</td>
<td>112.6</td>
<td>7.9</td>
</tr>
</tbody>
</table>

Table 3: CPU Usage (with dedicated compute servers) – CPU time (in minutes) of different architectures (normalized to the time of 1 vCPU).

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Pullup</th>
<th>PD-only</th>
<th>CA-only</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>249.6</td>
<td>48.5</td>
<td>70.3</td>
<td>23.2</td>
</tr>
<tr>
<td>Storage</td>
<td>0.0</td>
<td>31.1</td>
<td>0.0</td>
<td>7.4</td>
</tr>
<tr>
<td>Total</td>
<td>249.6</td>
<td>79.6</td>
<td>70.3</td>
<td>30.6</td>
</tr>
</tbody>
</table>
Pushdown DBMS – Q/A

Why weighted LFU instead of LRU?
Idea applied to real-world applications?
Any drawbacks or limitations of FPDB?
How scalable FPDB is?
Do pushdown mechanisms work for OLTP workloads?
How to balance the tradeoff between storage-layer computation cost and network reduction?
How to adapt query optimizer to different pushdown layers?
What operators to push down vs. stay in compute nodes?
Next Lecture

Anil Shanbhag, et al., A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. SIGMOD, 2020

Submit course evaluation on aefis.wisc.edu