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_J1PMidaAu/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

Xinyue Yu1, Min You2, Matthew Wozniak3, Abhishek Ghan CN23, Mario Serfati4, Ashraf Abdallah5, Michael Stonebraker6

1University of Wisconsin-Madison 2Yahoo Research 3University of Massachusetts Amherst 4University of Colorado Boulder 5University of Massachusetts at Amherst 6Stanford University

Email: yuxinyue@cs.wisc.edu, minyou@cs.uw.edu, mwozniak@cs.umass.edu, asalhab@buffalo.edu, mstonebru@cs.stanford.edu

Today’s Papers

ICDE 2020

FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS

Yilei Yang1, Matt You2, Matthew Wozniak3, Yilun Liu3, Xinyue Yu3, Mario Serfati4, Abhishek Ghan CN23, Michael Stonebraker6

1University of Wisconsin-Madison 2Yahoo Research 3University of Massachusetts Amherst 4University of Colorado Boulder 5University of Massachusetts at Amherst 6Stanford University

Email: yyang@cs.wisc.edu, myou@cs.uw.edu, mwozniak@cs.umass.edu, aasalhab@buffalo.edu, mstonebru@cs.stanford.edu

Abstract

Modern cloud databases adopt a compute-agnostic architecture that separates the management of computation and storage. A key bottleneck in such an environment is the network connecting the computation and storage layers. Two solutions have been explored to mitigate the back-and-forth caching and computation problems. While both techniques can significantly reduce network traffic, existing work considers them in orthogonal manners and support only one or the other, leaving potential performance benefits unexploited. We introduce a new hybrid system that uses both caching and computation as a unified, compute-agnostic architecture. This is the first comprehensive study of pushing pushdown computation into a cloud database. Our system provides pushdown and caching capabilities as first-class citizens, allowing users to express complex queries using standard SQL. We implemented our system on the Amazon Web Services Cloud Database (AWS CDB) platform, which offers high-performance S3 storage and cost-effective caching. We show that our hybrid system significantly improves query performance.

1 Introduction

Database and cloud management services are exposed as a single unified service. Users can access these services through a rich set of RESTful APIs that allow for a variety of end-to-end applications. However, the separation of these services results in increased network latency and increased costs. Our system provides a new way to improve the performance of network-transparent cloud services.

ICDE 2020

VLDB 2021

3
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
PushdownDB Architecture

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
S3 Select

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
Filter

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

```
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
```
Evaluation – All Operators and TPC-H

Overall, PushdownDB reduces runtime by 6.7× and reduces cost by 30%
PushdownDB: Accelerating a DBMS Using S3 Computation

Xinyao Yu1, Ming You2, Matthew You3, Abhiman Shankar4, Miao Shen5, Ashraf Aboualig6, Michael Stonebraker7

1. Department of Computer Science, University of Washington, Seattle, WA 98195, USA
2. Alibaba Cloud, Hangzhou, Zhejiang, China
3. Alibaba Cloud, Hangzhou, Zhejiang, China
4. Alibaba Cloud, Hangzhou, Zhejiang, China
5. Alibaba Cloud, Hangzhou, Zhejiang, China
6. Alibaba Cloud, Hangzhou, Zhejiang, China
7. Alibaba Cloud, Hangzhou, Zhejiang, China

ABSTRACT

We describe a novel architecture for a database management system that combines the storage and computation layers in a cloud environment. The system uses an S3-based storage layer to store and manage data, while computation is performed on S3 objects using a novel pushdown compiler that transforms SQL queries into S3 compute plans. The S3 compute plans are then executed on S3 to efficiently perform a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

1. INTRODUCTION

The immense growth of data in the cloud has led to a proliferation of cloud-based computing and storage services. This growth has led to the development of a new class of database management systems (DBMS) that are designed to operate in a cloud environment. These systems are characterized by their ability to store and process large volumes of data, and their ability to scale to meet the needs of large-scale cloud applications.

PushdownDB is a cloud-based DBMS that uses a novel architecture to combine the storage and computation layers. The system is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

2. DATA MANAGEMENT IN THE CLOUD

Cloud providers such as AWS offer a wide variety of compute instances (i.e., EC2, Elastic Compute Cloud) and storage services (i.e., S3, Elastic Block Store, EBS) for running and storing applications. However, some cloud environments have limitations on the type of data that can be stored or processed on the server. In such cases, applications may need to resort to using S3 as a primary storage layer. In this work, we show how to build an S3-based DBMS that can efficiently perform a wide range of operations, including join, aggregate, group, and sort.

3. The S3 Pushdown Compiler

The S3 pushdown compiler performs query optimization and execution in S3. It takes an S3 query as input and generates an S3 compute plan, which is then executed on S3 to perform the requested operations. The S3 pushdown compiler is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

4. Conclusion

In this work, we have described a novel architecture for a cloud-based DBMS that combines the storage and computation layers. The system is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS

Yeye Yang1, Matt You2, Matthew You3, Yihua Li4, Xinyao Yu5, Miao Shen6, Ashraf Aboualig7, Michael Stonebraker8

1. Department of Computer Science, University of Washington, Seattle, WA 98195, USA
2. Alibaba Cloud, Hangzhou, Zhejiang, China
3. Alibaba Cloud, Hangzhou, Zhejiang, China
4. Alibaba Cloud, Hangzhou, Zhejiang, China
5. Alibaba Cloud, Hangzhou, Zhejiang, China
6. Alibaba Cloud, Hangzhou, Zhejiang, China
7. Alibaba Cloud, Hangzhou, Zhejiang, China
8. Alibaba Cloud, Hangzhou, Zhejiang, China

ABSTRACT

FlexPushdownDB is a hybrid pushdown and caching architecture for a cloud database that combines the storage and computation layers. The system is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

1. INTRODUCTION

Cloud-based database systems are a common architecture for cloud-based applications. However, these systems can suffer from scalability and performance issues when dealing with large volumes of data. To address these issues, we propose a hybrid pushdown and caching architecture for cloud-based databases. The system is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

2. DATA MANAGEMENT IN THE CLOUD

Cloud providers such as AWS offer a wide variety of compute instances (i.e., EC2, Elastic Compute Cloud) and storage services (i.e., S3, Elastic Block Store, EBS) for running and storing applications. However, some cloud environments have limitations on the type of data that can be stored or processed on the server. In such cases, applications may need to resort to using S3 as a primary storage layer. In this work, we show how to build an S3-based DBMS that can efficiently perform a wide range of operations, including join, aggregate, group, and sort.

3. The S3 Pushdown Compiler

The S3 pushdown compiler performs query optimization and execution in S3. It takes an S3 query as input and generates an S3 compute plan, which is then executed on S3 to perform the requested operations. The S3 pushdown compiler is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.

4. Conclusion

In this work, we have described a novel architecture for a cloud-based DBMS that combines the storage and computation layers. The system is designed to be highly scalable and efficient, and it is capable of performing a wide range of operations, including join, aggregate, group, and sort. This approach allows for highly scalable and cost-effective data processing in a cloud environment.
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.
Mitigate Network Bottleneck

**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
FlexPushdownDB (FPDB) Overview

Design choices
– Cache table data rather than query results for simplicity

Source code: https://github.com/cloud-olap/FlexPushdownDB
Design choices
– Cache table data rather than query results for simplicity
– Segment as the caching granularity

Employee

<table>
<thead>
<tr>
<th>ID</th>
<th>Age</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Age</td>
<td>...</td>
</tr>
<tr>
<td>ID</td>
<td>Age</td>
<td>...</td>
</tr>
</tbody>
</table>

Partition 1

Partition 2

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

Employee

Partition 1

Segment

Partition 2

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

Main modules

- Hybrid executor
- Cache manager
  - Admission
  - Eviction

Query plan

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.
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>
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 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