WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

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/1Re1M9FmJwI_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

Today's Papers

PushdownDB: Accelerating a DBMS Using S3 Computation

Xiangyao Yu*, Matt Youill‡, Matthew Woicik†, Abdurrahman Ghanem§, Marco Serafini¶, Ashraf Aboulnaga§, Michael Stonebraker†
*University of Wisconsin-Madison †Massachusetts Institute of Technology
‡Burnian §Qatar Computing Research Institute ¶University of Massachusetts Amherst
Email: yxy@cs.wisc.edu, matt.youill@burnian.com, mwoicik@mit.edu, abghanem@hbku.edu.qa, marco@cs.umass.edu, aaboulnaga@hbku.edu.qa, stonebraker@csail.mit.edu

Abstract—**This paper studies the effectiveness of pushing parts of DBMS analytics queries into the Simple Storage Service (S3) of Amazon Web Services (AWS), using a recently released capability called S3 Select. We show that some DBMS primitives (filter, projection, and aggregation) can always be cost-effectively moved into S3. Other more complex operations (join, top-K, and group-by) require reimplementation to take advantage of S3 Select and are often candidates for pushdown. We demonstrate these capabilities through experimentation using a new DBMS that we developed, *PushdownDB*. Experimentation with a collection of queries including TPC-H queries shows that *PushdownDB* is on average 30% cheaper and 6.7× faster than a baseline that does not use S3 Select.**

I. Introduction

Clouds offer cheaper and more flexible computing than "on-prem". Not only can one add resources on the fly, the large cloud vendors have major economies of scale relative to "on-prem" deployment. Modern clouds employ an architecture where the computation and storage are disaggregated — the two components are independently managed and connected using a network. Such an architecture allows for independent scaling of computation and storage, which simplifies the management of storage and reduces its cost. A number of data warehousing systems have been built to analyze data on disaggregated cloud storage, including Presto [1], Snowflake [2], Redshift Spectrum [3], among others.

In a disaggregated architecture, the network that connects the computation and storage layers can be a major performance bottleneck. Two intuitive solutions are *caching* and *computation pushdown*. With caching, a compute server loads data from the remote storage and caches it in main memory or local storage, amortizing the network transfer cost. Caching has been implemented in Snowflake [2] and Redshift Spectrum [3], [4]. With computation pushdown, a database management system (DBMS) pushes its functionality as close to storage as possible. Previous research [5] and systems (e.g., Britton-Lee IDM 500 [6], Oracle Exadata server [7], and IBM Netezza machine [8]) have shown that this can significantly improve performance.

Recently, Amazon Web Services (AWS) introduced a feature called "S3 Select", through which limited computation can be pushed onto their shared cloud storage service called S3 [9]. This provides an opportunity to revisit the question of how to divide query processing tasks between S3 storage nodes and normal computation nodes. The question is nontrivial as the limited computational interface of S3 Select allows only certain simple 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 pricing can be more expensive than computing on normal EC2 nodes.

In this paper, we set our goal to understand the performance of computation pushdown when running queries in a cloud setting with disaggregated storage. Specifically, we consider filter (with and without indexing), join, group-by, and top-K as candidates. We implement these operators to take advantage of computation pushdown through S3 Select and study their cost and performance. We show dramatic performance improvement and cost reduction, even with the relatively high cost of S3 Select. In addition, we analyze queries from the TPC-H benchmark and show similar benefits of performance and cost. We point out the limitations of the current S3 Select service and provide several suggestions based on the lessons we learned from this project. To the best of our knowledge, this is the *first extensive study of pushdown computing for database operators in a disaggregated architecture.* A more detailed description of this work can be found in [10].

II. Data Management in the Cloud

Cloud providers such as AWS offer a wide variety of computing instances (i.e., EC2: Elastic Compute Cloud) and storage services (i.e., EBS: Elastic Block Store, EFS: Elastic File System, and S3: Simple Storage Service). Compared to other storage services, S3 is a highly available object store that provides virtually infinite storage capacity for regular users with relatively low cost, and is supported by many popular cloud databases, including Presto [1], Hive [11], Spark SQL [12], Redshift Spectrum [3], and Snowflake [2]. The storage nodes in S3 are separate from compute nodes. Hence, a DBMS uses S3 as a storage system and transfers needed data over a network for query processing.

To reduce network traffic and the associated processing on compute nodes, AWS released a new service called *S3 Select* [9] in 2018 to push limited computation to the storage nodes. At the current time, S3 Select supports only selection,

ICDE 2020

FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS

Yifei Yang[1], Matt Youill[2], Matthew Woicik[3], Yizhou Liu[1], Xiangyao Yu[1], Marco Serafini[4], Ashraf Aboulnaga[5], Michael Stonebraker[3]
[1]University of Wisconsin-Madison, [2]Burnian, [3]Massachusetts Institute of Technology, [4]University of Massachusetts-Amherst, [5]Qatar Computing Research Institute
[1]{yyang673@, liu773@, yxy@cs.}wisc.edu, [2]matt.youill@burnian.com, [3]{mwoicik@, stonebraker@csail.}mit.edu, [4]marco@cs.umass.edu, [5]aaboulnaga@hbku.edu.qa

ABSTRACT

Modern cloud databases adopt a *storage-disaggregation* architecture that separates the management of computation and storage. A major bottleneck in such an architecture is the network connecting the computation and storage layers. Two solutions have been explored to mitigate the bottleneck: *caching* and *computation pushdown*. While both techniques can significantly reduce network traffic, existing DBMSs consider them as orthogonal techniques and support only one or the other, leaving potential performance benefits unexploited.

In this paper we present *FlexPushdownDB* (*FPDB*), an OLAP cloud DBMS prototype that supports fine-grained hybrid query execution to combine the benefits of caching and computation pushdown in a storage-disaggregation architecture. We build a hybrid query executor based on a new concept called *separable operators* to combine the data from the cache and results from the pushdown processing. We also propose a novel *Weighted-LFU* cache replacement policy that takes into account the cost of pushdown computation. Our experimental evaluation on the Star Schema Benchmark shows that the hybrid execution outperforms both the conventional *caching-only* architecture and *pushdown-only* architecture by 2.2×. In the hybrid architecture, our experiments show that Weighted-LFU can outperform the baseline LFU by 37%.

PVLDB Reference Format:
Yifei Yang, Matt Youill, Matthew Woicik, Yizhou Liu, Xiangyao Yu, Marco Serafini, Ashraf Aboulnaga, Michael Stonebraker. FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. PVLDB, 14(11): 2101 - 2113, 2021.
doi:10.14778/3476249.3476265

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/cloud-olap/FlexPushdownDB.git.

1 INTRODUCTION

Database management systems (DBMSs) are gradually moving from on-premises to the cloud for higher elasticity and lower cost. Modern cloud DBMSs adopt a *storage-disaggregation architecture* that divides computation and storage into separate layers of servers connected through the network, simplifying provisioning and enabling independent scaling of resources. However, disaggregation requires rethinking a fundamental principle of distributed DBMSs: "move computation to data rather than data to computation". Compared to the traditional shared-nothing architecture, which embodies that principle and stores data on local disks, the network in the disaggregation architecture typically has lower bandwidth than local disks, making it a potential performance bottleneck.

Two solutions have been explored to mitigate this network bottleneck: *caching* and *computation pushdown*. Both solutions can reduce the amount of data transferred between the two layers. Caching keeps the hot data in the computation layer. Examples include Snowflake [21, 48] and Presto with Alluxio cache service [14]. The Redshift [30] layer in Redshift Spectrum [8] can also be considered as a cache with user-controlled contents. With computation pushdown, filtering and aggregation are performed close to the storage with only the results returned. Examples include Oracle Exadata [49], IBM Netezza [23], AWS Redshift Spectrum [8], AWS Aqua [12], and PushdownDB [53]. The fundamental reasons that caching and pushdown have performance benefits are that local memory and storage have higher bandwidth than the network and that the internal bandwidth within the storage layer is also higher than that of the network.

Existing DBMSs consider caching and computation pushdown as *orthogonal*. Most systems implement only one of them. Some systems, such as Exadata [49], Netezza [23], Redshift Spectrum [8], and Presto [14] consider the two techniques as independent: query operators can either access cached data (i.e., full tables) or push down computation on remote data, but not both.

In this paper, we argue that caching and computation pushdown are *not* orthogonal techniques, and that the rigid dichotomy of existing systems leaves potential performance benefits unexploited. We propose *FlexPushdownDB* (*FPDB* in short), an OLAP cloud DBMS prototype that combines the benefits of caching and pushdown.

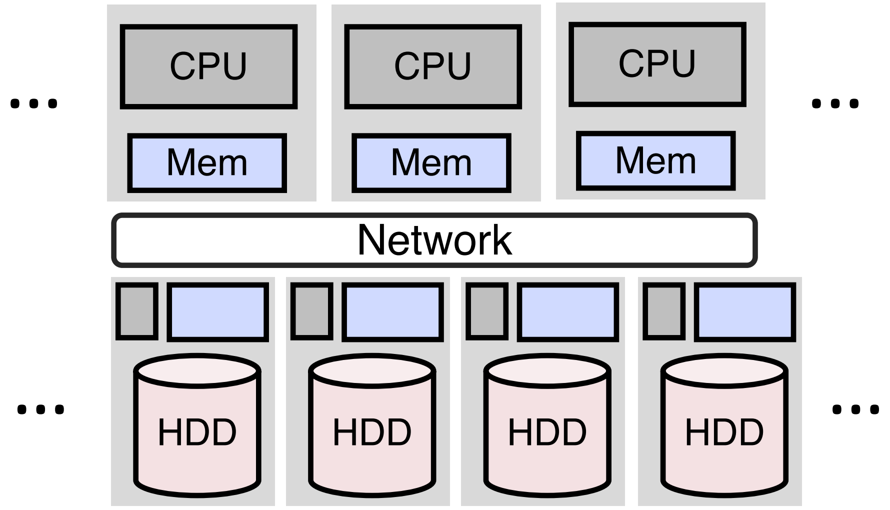
FPDB introduces the concept of *separable operators*, which combine local computation on cached segments and pushdown on the segments in the cloud storage. This hybrid execution can leverage cached data at a fine granularity. While not all relational operators are separable, some of the most commonly-used ones are, including filtering, projection, aggregation. We introduce a *merge* operator to combine the outputs from caching and pushdown.

Separable operators open up new possibilities for caching. Traditional cache replacement policies assume that each miss requires

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476265

VLDB 2021

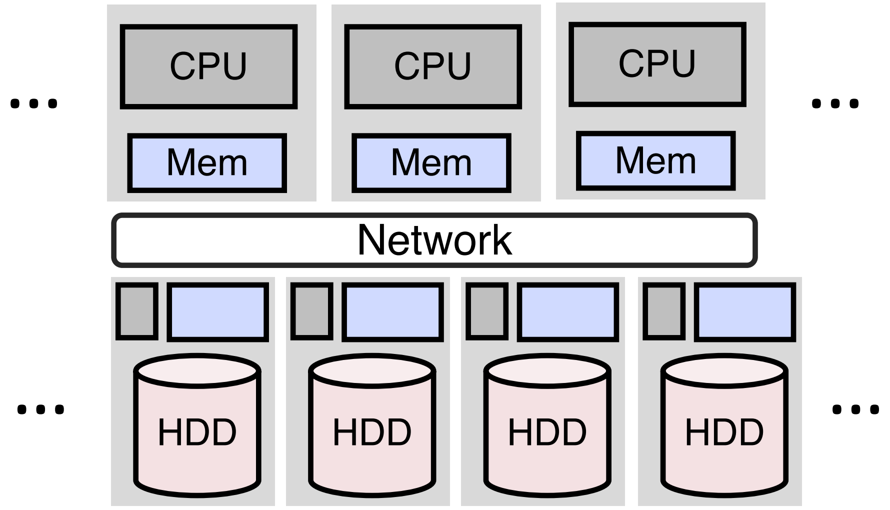
Storage-Disaggregation Architecture



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

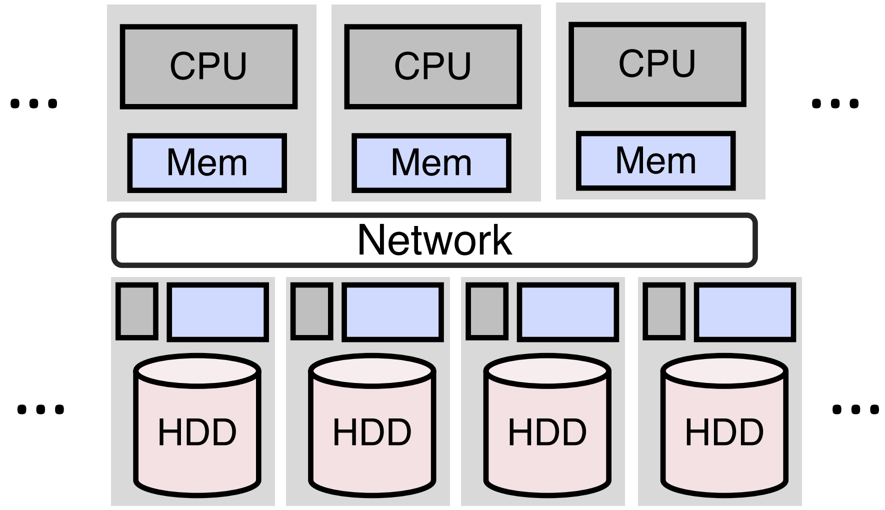
Advantages

- Lower management cost
- Independent scaling of computation and storage

Disadvantages

- **Network becomes a bottleneck**

How to Mitigate the Network Bottleneck?



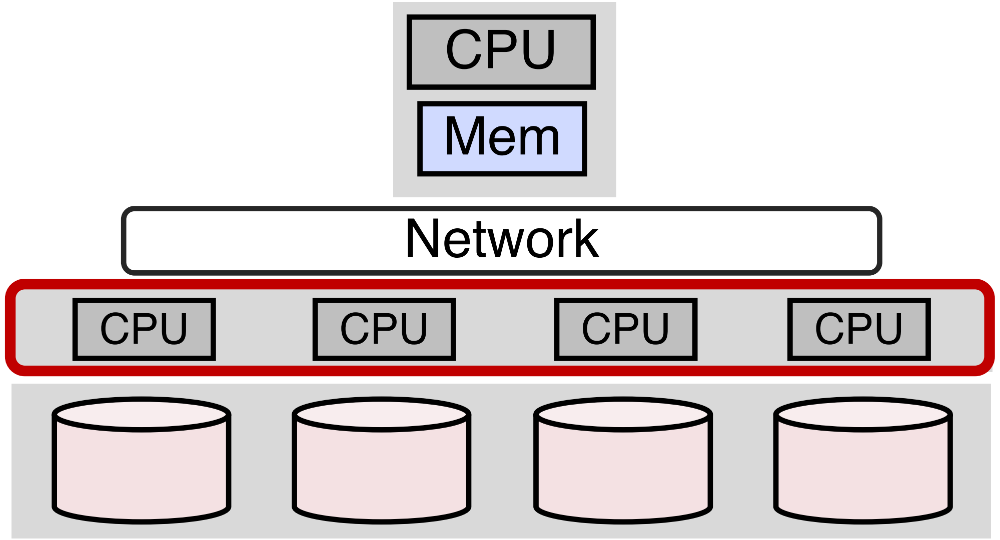
Solution 1: Move data to computation

- Cache storage data in the computation layer
- Example: Snowflake

Solution 2: **Move computation to data**

- Pushdown computation to the storage layer
- Example: PushdownDB

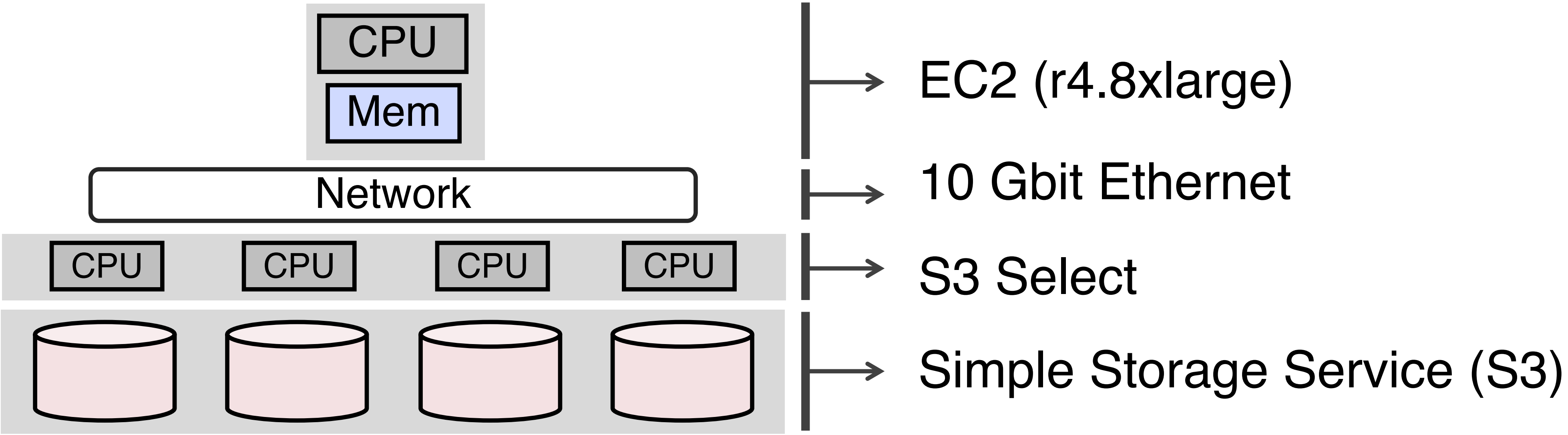
PushdownDB Architecture



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

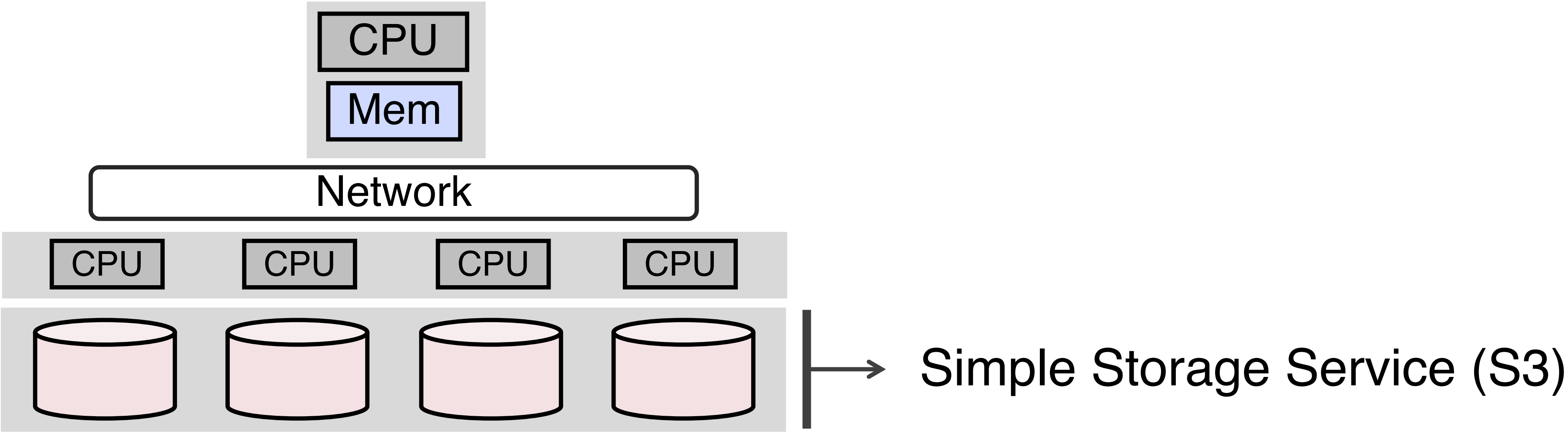


PushdownDB implementation

- 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)

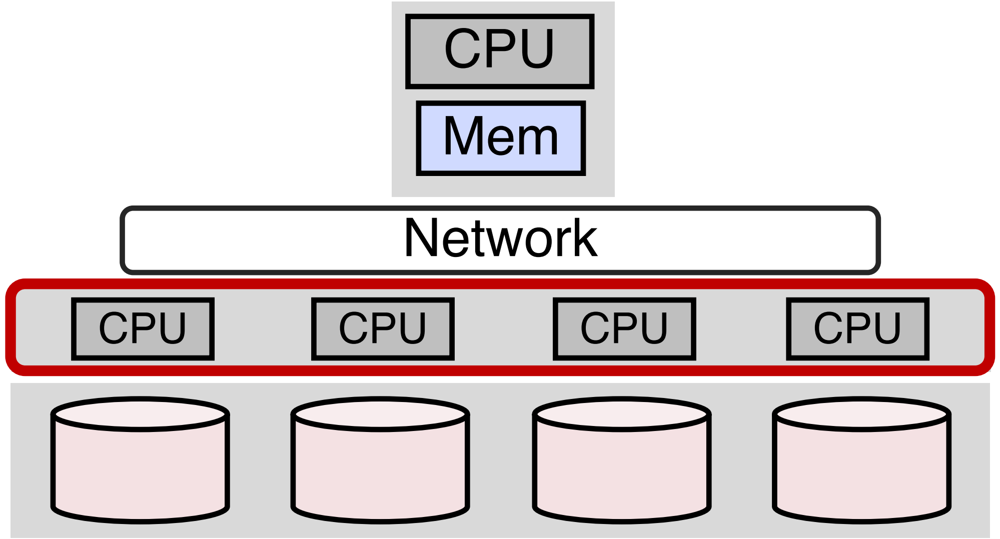


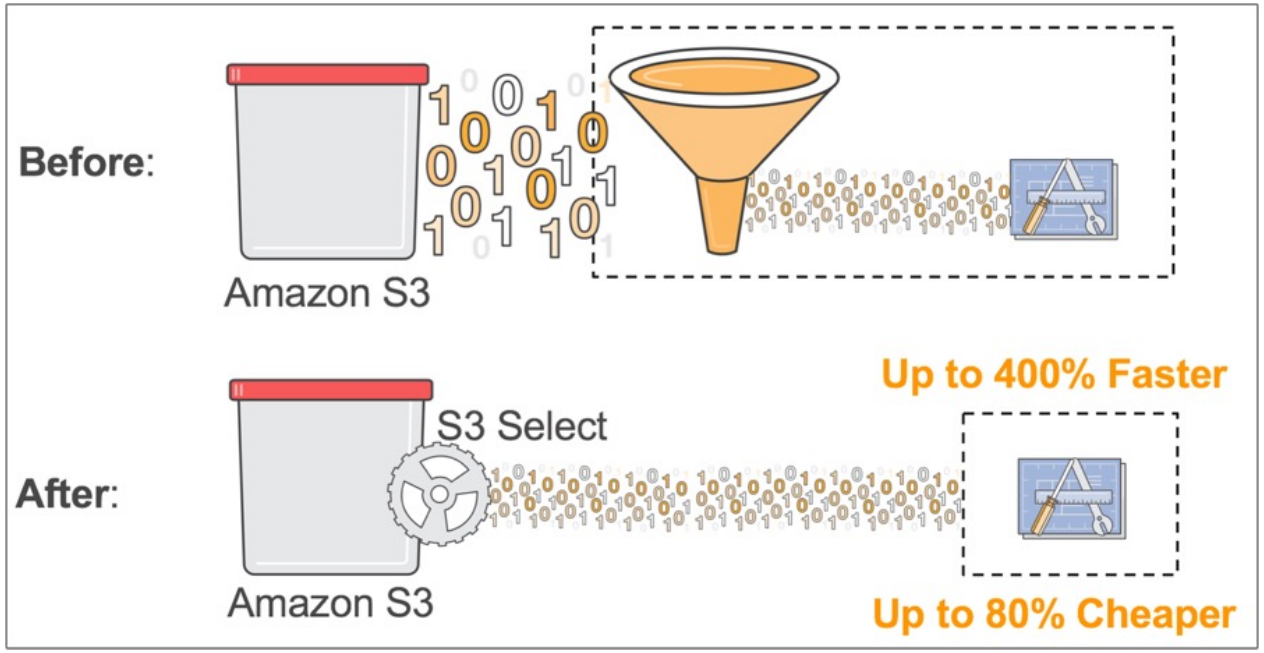
Virtually infinite storage capacity with relatively low cost

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**

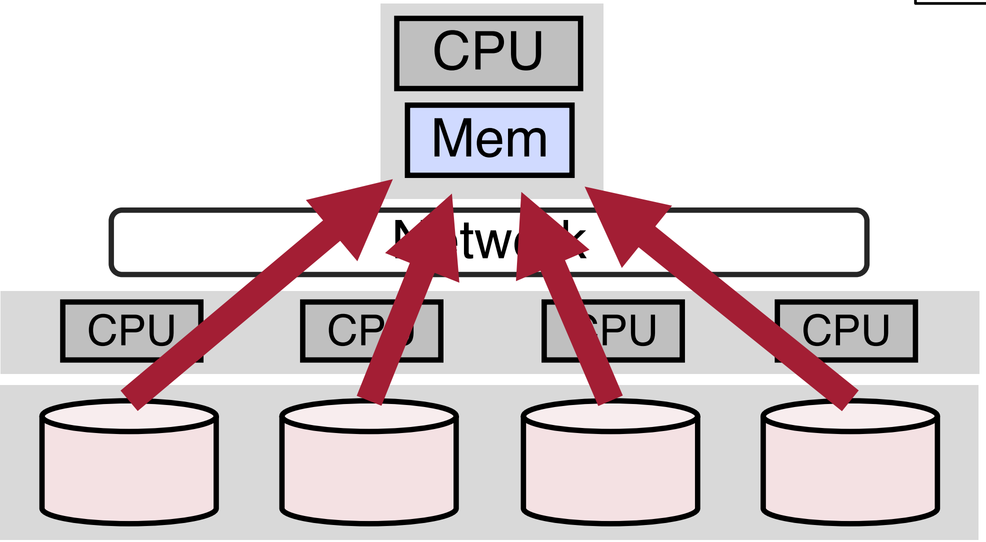
Filter

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

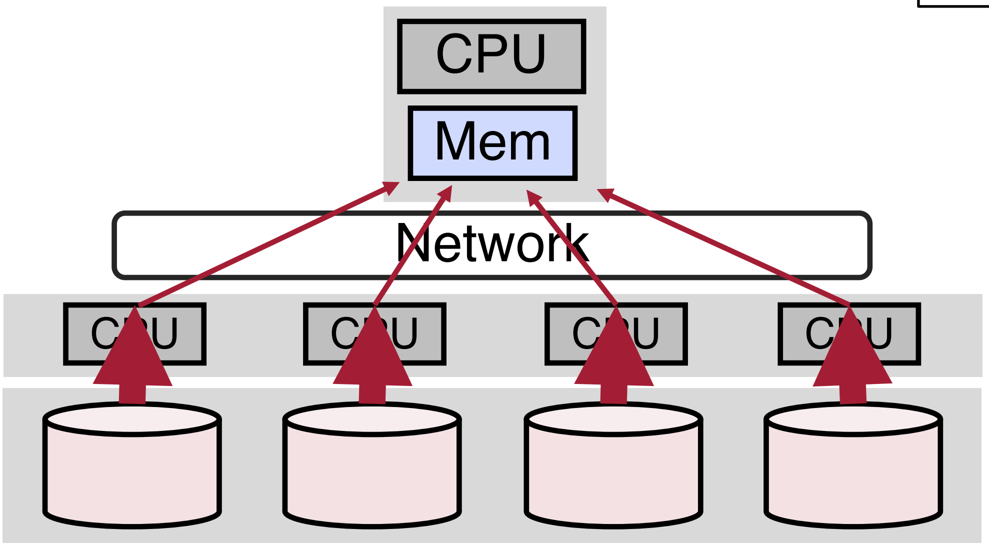
– Compute server loads entire table from S3 and filters locally

S3-side filtering

– Push down predicate evaluation using S3 Select

Example query:

```
SELECT col1, col2
FROM R
WHERE col1 < 10
```



Join

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
```

Join

Bloom Join

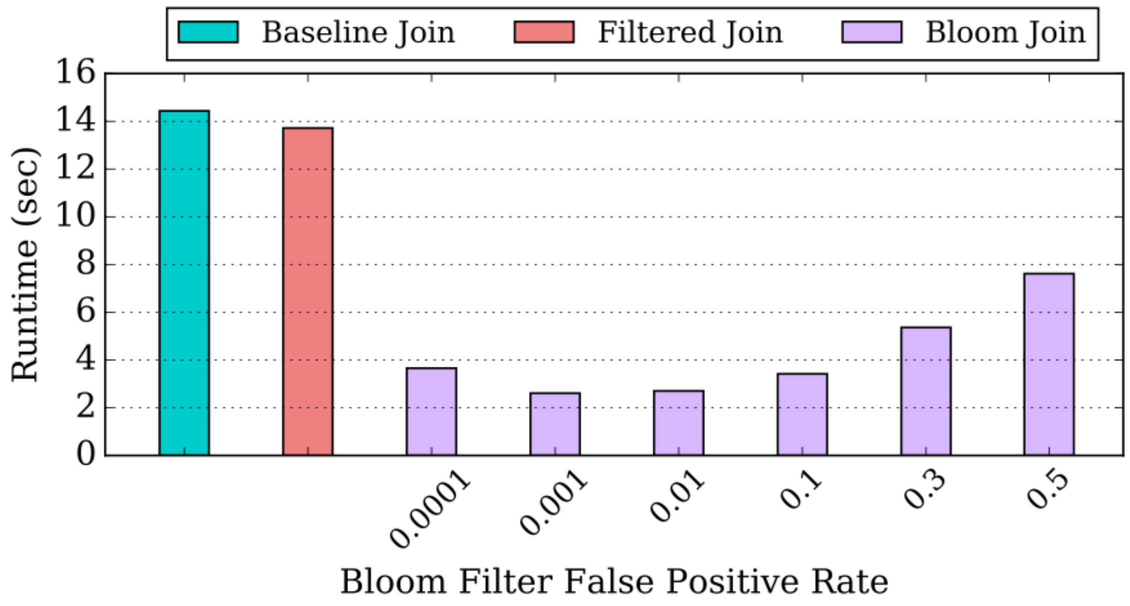
- Step 1: Server loads the smaller table, builds a bloom filter using join key
- Step 2: Server sends the filter via S3 Select to load the bigger table
- Bloom filter is pushed down as a predicate

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

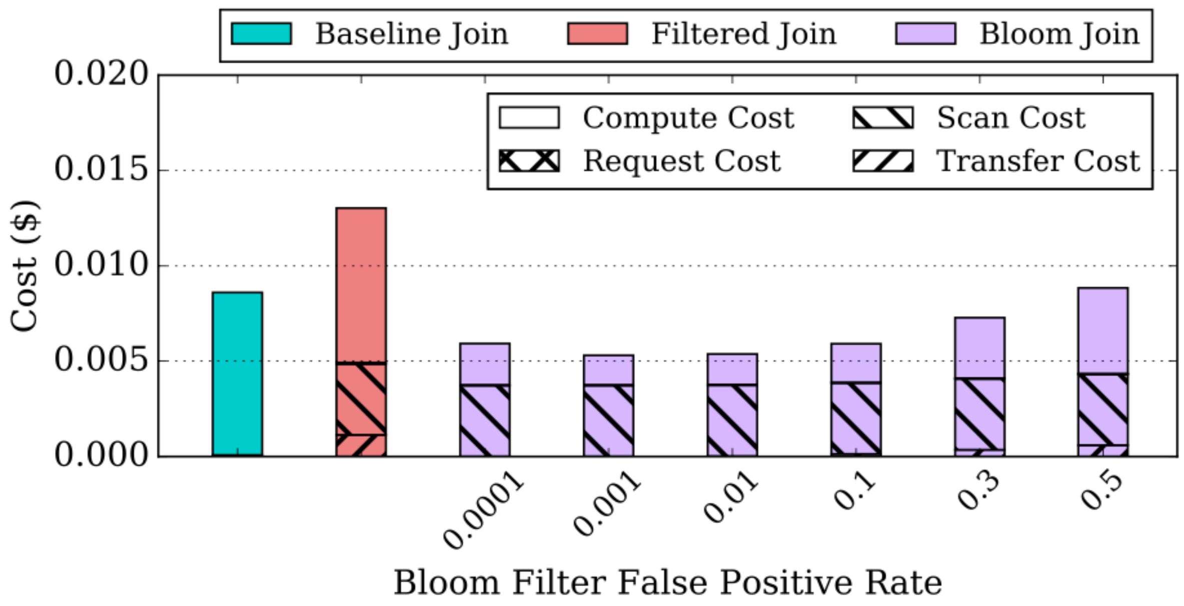
```
SELECT SUM(O_TOTALPRICE)
FROM CUSTOMER, ORDER
WHERE
    O_CUSTKEY = C_CUSTKEY
    AND C_ACCTBAL <= upper_c_acctbal
    AND O_ORDERDATE < upper_o_orderdate
```

Evaluation – Join

Runtime

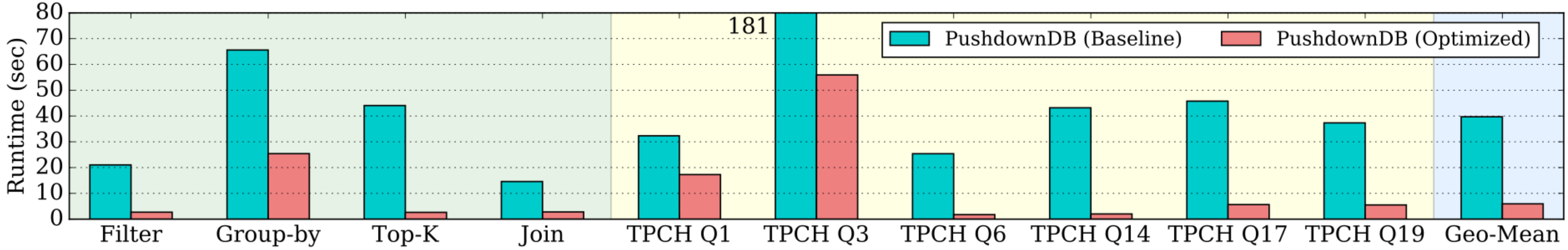


Cost Breakdown

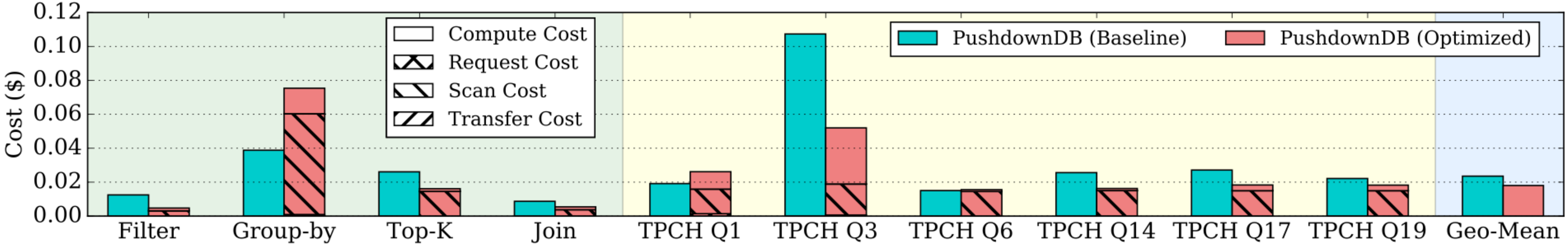



```
SELECT SUM(O_TOTALPRICE)
FROM CUSTOMER, ORDER
WHERE
      O_CUSTKEY = C_CUSTKEY
      AND C_ACCTBAL <= upper_c_acctbal
      AND O_ORDERDATE < upper_o_orderdate
```

Evaluation – All Operators and TPC-H



(a) Runtime



(b) Cost

Overall, PushdownDB **reduces runtime by 6.7×** and **reduces cost by 30%**

Today's Papers

PushdownDB: Accelerating a DBMS Using S3 Computation

Xiangyao Yu*, Matt Youill‡, Matthew Woicik†, Abdurrahman Ghanem§,
Marco Serafini¶, Ashraf Aboulnaga§, Michael Stonebraker†
*University of Wisconsin-Madison †Massachusetts Institute of Technology
‡Burnian §Qatar Computing Research Institute ¶University of Massachusetts Amherst
Email: yxy@cs.wisc.edu, matt.youill@burnian.com, mwoicik@mit.edu, abghanem@hbku.edu.qa,
marco@cs.umass.edu, aaboulnaga@hbku.edu.qa, stonebraker@csail.mit.edu

***Abstract*—This paper studies the effectiveness of pushing parts of DBMS analytics queries into the Simple Storage Service (S3) of Amazon Web Services (AWS), using a recently released capability called S3 Select. We show that some DBMS primitives (filter, projection, and aggregation) can always be cost-effectively moved into S3. Other more complex operations (join, top-K, and group-by) require reimplementation to take advantage of S3 Select and are often candidates for pushdown. We demonstrate these capabilities through experimentation using a new DBMS that we developed, *PushdownDB*. Experimentation with a collection of queries including TPC-H queries shows that *PushdownDB* is on average 30% cheaper and 6.7× faster than a baseline that does not use S3 Select.**

I. Introduction

Clouds offer cheaper and more flexible computing than "on-prem". Not only can one add resources on the fly, the large cloud vendors have major economies of scale relative to "on-prem" deployment. Modern clouds employ an architecture where the computation and storage are disaggregated — the two components are independently managed and connected using a network. Such an architecture allows for independent scaling of computation and storage, which simplifies the management of storage and reduces its cost. A number of data warehousing systems have been built to analyze data on disaggregated cloud storage, including Presto [1], Snowflake [2], Redshift Spectrum [3], among others.

In a disaggregated architecture, the network that connects the computation and storage layers can be a major performance bottleneck. Two intuitive solutions are *caching* and *computation pushdown*. With caching, a compute server loads data from the remote storage and caches it in main memory or local storage, amortizing the network transfer cost. Caching has been implemented in Snowflake [2] and Redshift Spectrum [3], [4]. With computation pushdown, a database management system (DBMS) pushes its functionality as close to storage as possible. Previous research [5] and systems (e.g., Britton-Lee IDM 500 [6], Oracle Exadata server [7], and IBM Netezza machine [8]) have shown that this can significantly improve performance.

Recently, Amazon Web Services (AWS) introduced a feature called "S3 Select", through which limited computation can be pushed onto their shared cloud storage service called S3 [9]. This provides an opportunity to revisit the question of how to divide query processing tasks between S3 storage nodes and normal computation nodes. The question is nontrivial as the limited computational interface of S3 Select allows only certain simple 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 pricing can be more expensive than computing on normal EC2 nodes.

In this paper, we set our goal to understand the performance of computation pushdown when running queries in a cloud setting with disaggregated storage. Specifically, we consider filter (with and without indexing), join, group-by, and top-K as candidates. We implement these operators to take advantage of computation pushdown through S3 Select and study their cost and performance. We show dramatic performance improvement and cost reduction, even with the relatively high cost of S3 Select. In addition, we analyze queries from the TPC-H benchmark and show similar benefits of performance and cost. We point out the limitations of the current S3 Select service and provide several suggestions based on the lessons we learned from this project. To the best of our knowledge, this is the *first extensive study of pushdown computing for database operators in a disaggregated architecture.* A more detailed description of this work can be found in [10].

II. Data Management in the Cloud

Cloud providers such as AWS offer a wide variety of computing instances (i.e., EC2: Elastic Compute Cloud) and storage services (i.e., EBS: Elastic Block Store, EFS: Elastic File System, and S3: Simple Storage Service). Compared to other storage services, S3 is a highly available object store that provides virtually infinite storage capacity for regular users with relatively low cost, and is supported by many popular cloud databases, including Presto [1], Hive [11], Spark SQL [12], Redshift Spectrum [3], and Snowflake [2]. The storage nodes in S3 are separate from compute nodes. Hence, a DBMS uses S3 as a storage system and transfers needed data over a network for query processing.

To reduce network traffic and the associated processing on compute nodes, AWS released a new service called *S3 Select* [9] in 2018 to push limited computation to the storage nodes. At the current time, S3 Select supports only selection,

FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS

Yifei Yang[1], Matt Youill[2], Matthew Woicik[3], Yizhou Liu[1],
Xiangyao Yu[1], Marco Serafini[4], Ashraf Aboulnaga[5], Michael Stonebraker[3]
[1]University of Wisconsin-Madison, [2]Burnian, [3]Massachusetts Institute of Technology, [4]University of Massachusetts-Amherst, [5]Qatar Computing Research Institute
[1]{yyang673@, liu773@, yxy@cs.}wisc.edu, [2]matt.youill@burnian.com, [3]{mwoicik@, stonebraker@csail.}mit.edu,
[4]marco@cs.umass.edu, [5]aaboulnaga@hbku.edu.qa

ABSTRACT

Modern cloud databases adopt a *storage-disaggregation* architecture that separates the management of computation and storage. A major bottleneck in such an architecture is the network connecting the computation and storage layers. Two solutions have been explored to mitigate the bottleneck: *caching* and *computation pushdown*. While both techniques can significantly reduce network traffic, existing DBMSs consider them as orthogonal techniques and support only one or the other, leaving potential performance benefits unexploited.

In this paper we present *FlexPushdownDB* (*FPDB*), an OLAP cloud DBMS prototype that supports fine-grained hybrid query execution to combine the benefits of caching and computation pushdown in a storage-disaggregation architecture. We build a hybrid query executor based on a new concept called *separable operators* to combine the data from the cache and results from the pushdown processing. We also propose a novel *Weighted-LFU* cache replacement policy that takes into account the cost of pushdown computation. Our experimental evaluation on the Star Schema Benchmark shows that the hybrid execution outperforms both the conventional *caching-only* architecture and *pushdown-only* architecture by 2.2×. In the hybrid architecture, our experiments show that Weighted-LFU can outperform the baseline LFU by 37%.

PVLDB Reference Format:
Yifei Yang, Matt Youill, Matthew Woicik, Yizhou Liu, Xiangyao Yu, Marco Serafini, Ashraf Aboulnaga, Michael Stonebraker. FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. PVLDB, 14(11): 2101 - 2113, 2021.
doi:10.14778/3476249.3476265

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/cloud-olap/FlexPushdownDB.git.

1 INTRODUCTION

Database management systems (DBMSs) are gradually moving from on-premises to the cloud for higher elasticity and lower cost. Modern cloud DBMSs adopt a *storage-disaggregation architecture* that divides computation and storage into separate layers of servers connected through the network, simplifying provisioning and enabling independent scaling of resources. However, disaggregation requires rethinking a fundamental principle of distributed DBMSs: "move computation to data rather than data to computation". Compared to the traditional shared-nothing architecture, which embodies that principle and stores data on local disks, the network in the disaggregation architecture typically has lower bandwidth than local disks, making it a potential performance bottleneck.

Two solutions have been explored to mitigate this network bottleneck: *caching* and *computation pushdown*. Both solutions can reduce the amount of data transferred between the two layers. Caching keeps the hot data in the computation layer. Examples include Snowflake [21, 48] and Presto with Alluxio cache service [14]. The Redshift [30] layer in Redshift Spectrum [8] can also be considered as a cache with user-controlled contents. With computation pushdown, filtering and aggregation are performed close to the storage with only the results returned. Examples include Oracle Exadata [49], IBM Netezza [23], AWS Redshift Spectrum [8], AWS Aqua [12], and PushdownDB [53]. The fundamental reasons that caching and pushdown have performance benefits are that local memory and storage have higher bandwidth than the network and that the internal bandwidth within the storage layer is also higher than that of the network.

Existing DBMSs consider caching and computation pushdown as *orthogonal*. Most systems implement only one of them. Some systems, such as Exadata [49], Netezza [23], Redshift Spectrum [8], and Presto [14] consider the two techniques as independent: query operators can either access cached data (i.e., full tables) or push down computation on remote data, but not both.

In this paper, we argue that caching and computation pushdown are *not* orthogonal techniques, and that the rigid dichotomy of existing systems leaves potential performance benefits unexploited. We propose *FlexPushdownDB* (*FPDB* in short), an OLAP cloud DBMS prototype that combines the benefits of caching and pushdown.

FPDB introduces the concept of *separable operators*, which combine local computation on cached segments and pushdown on the segments in the cloud storage. This hybrid execution can leverage cached data at a fine granularity. While not all relational operators are separable, some of the most commonly-used ones are, including filtering, projection, aggregation. We introduce a *merge* operator to combine the outputs from caching and pushdown.

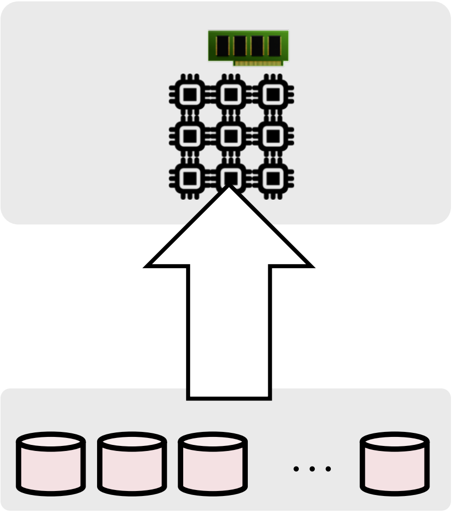
Separable operators open up new possibilities for caching. Traditional cache replacement policies assume that each miss requires

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476265

ICDE 2020

VLDB 2021

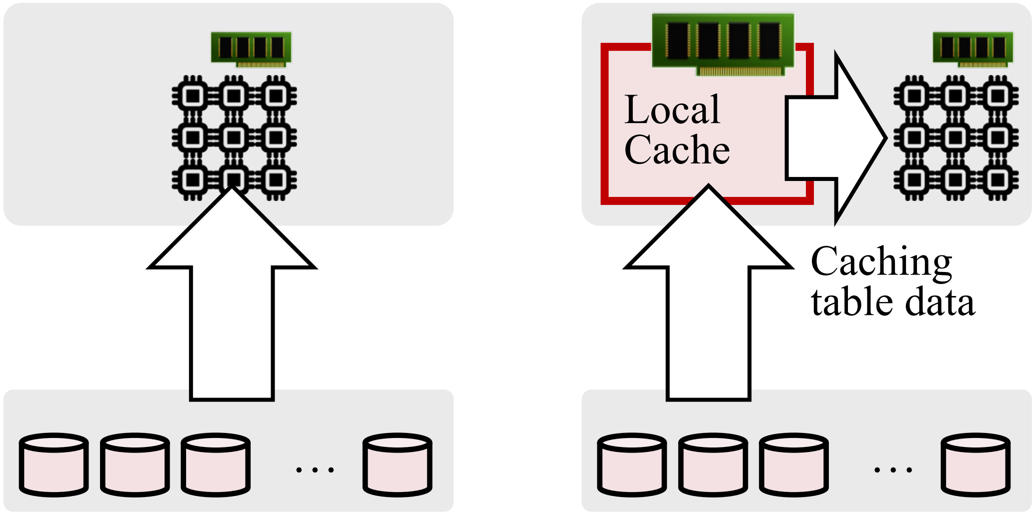
Mitigate Network Bottleneck



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

– Examples: default presto, hive, SparkSQL, etc.

Mitigate Network Bottleneck

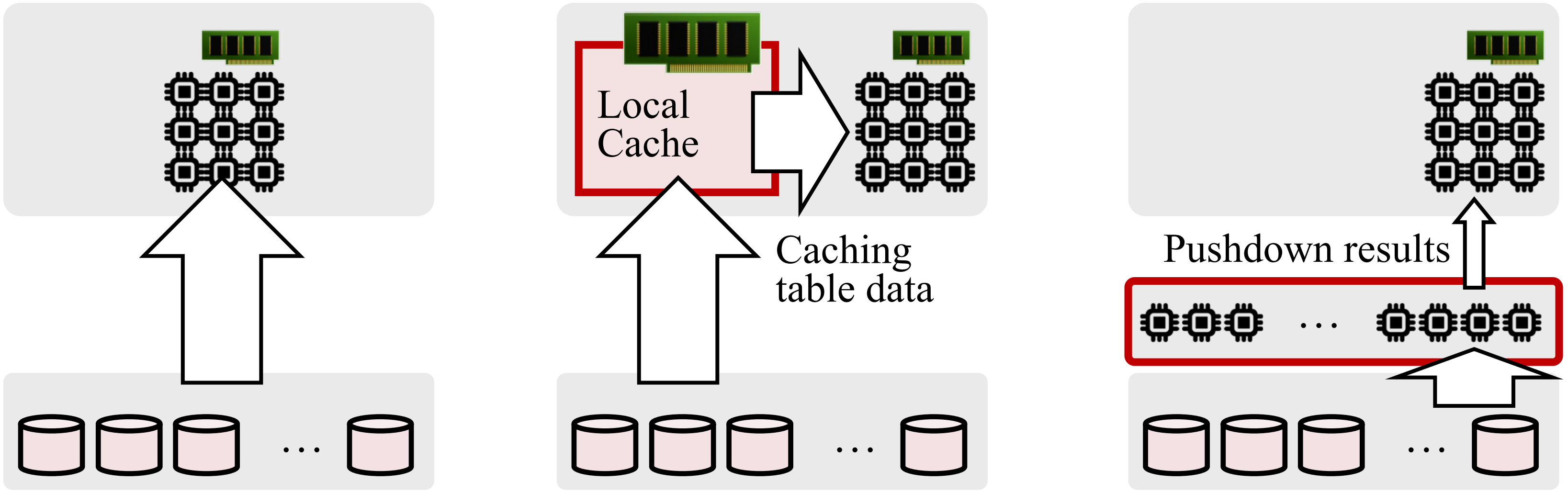


Baseline: always load data from cloud storage (e.g., S3)

Caching: cache hot table data in the compute node

– Examples: Snowflake, redshift spectrum (static), Alluxio, etc.

Mitigate Network Bottleneck



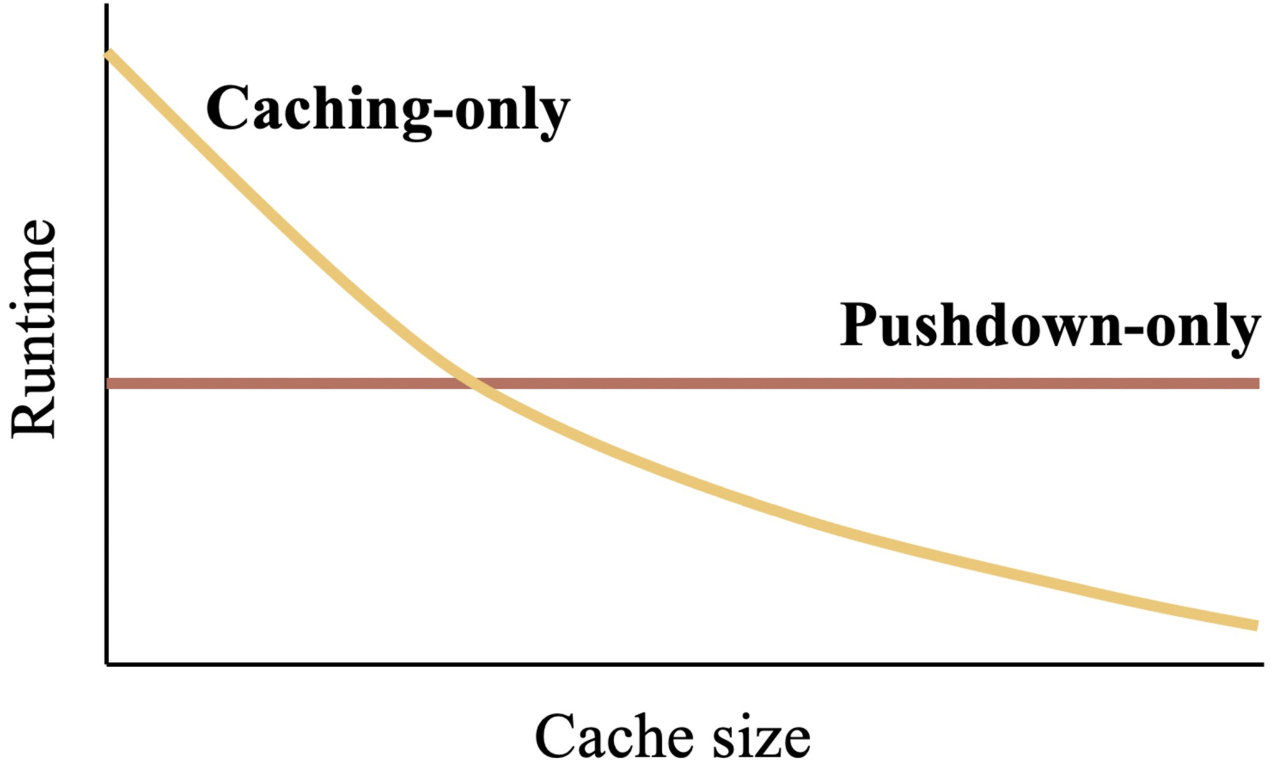
Baseline: always load data from cloud storage (e.g., S3)

Caching: cache hot table data in the compute node

Pushdown: push down selection, projection, aggregation to storage

– Examples: Redshift spectrum, Aqua, PushdownDB, etc.

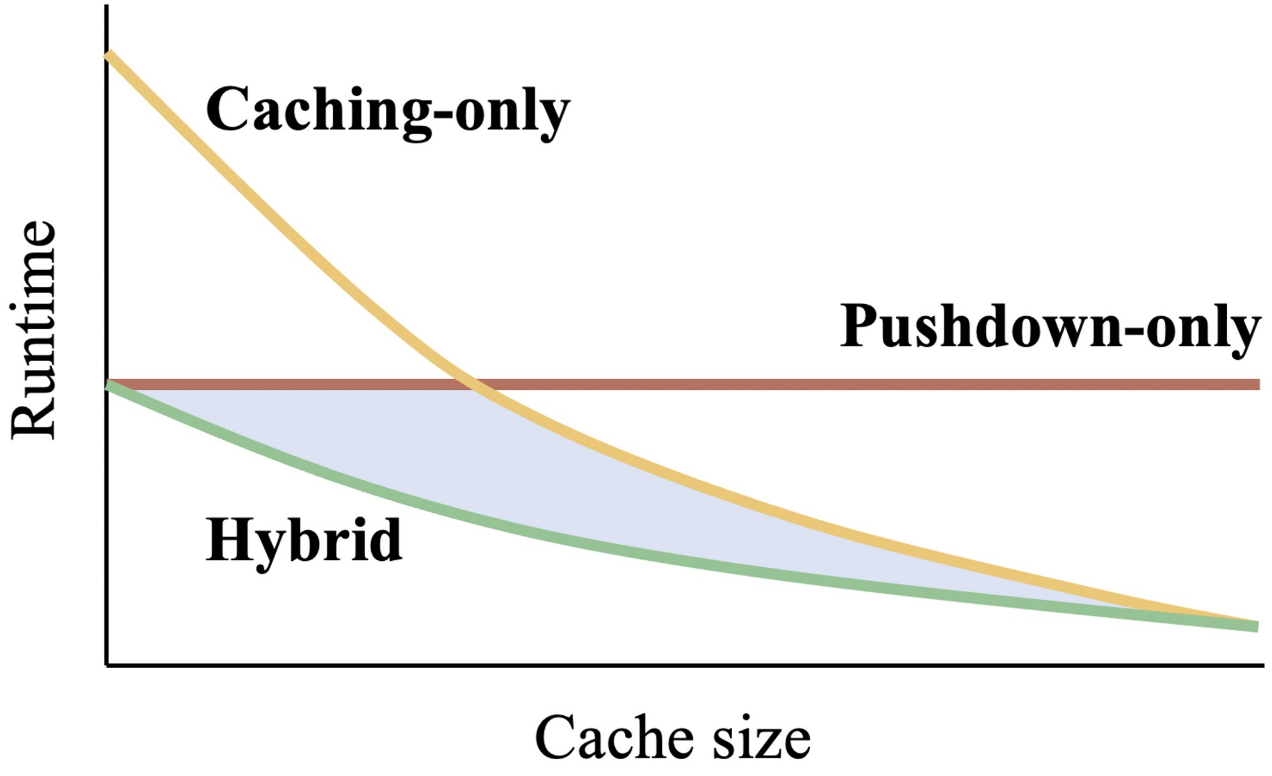
Caching vs. Pushdown



Caching performance increases with a bigger cache

Pushdown performance is independent of cache size

Caching vs. Pushdown

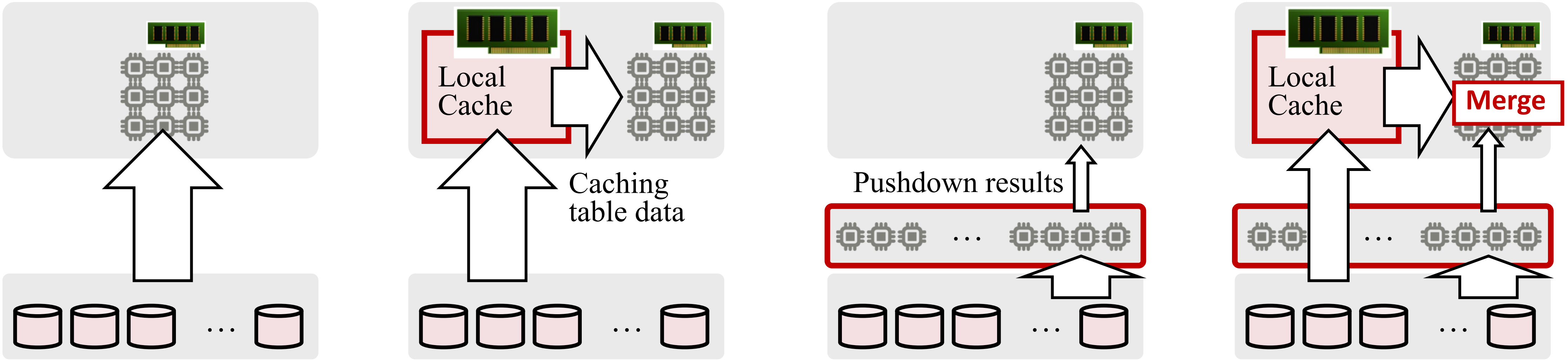


Caching performance increases with a bigger cache

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

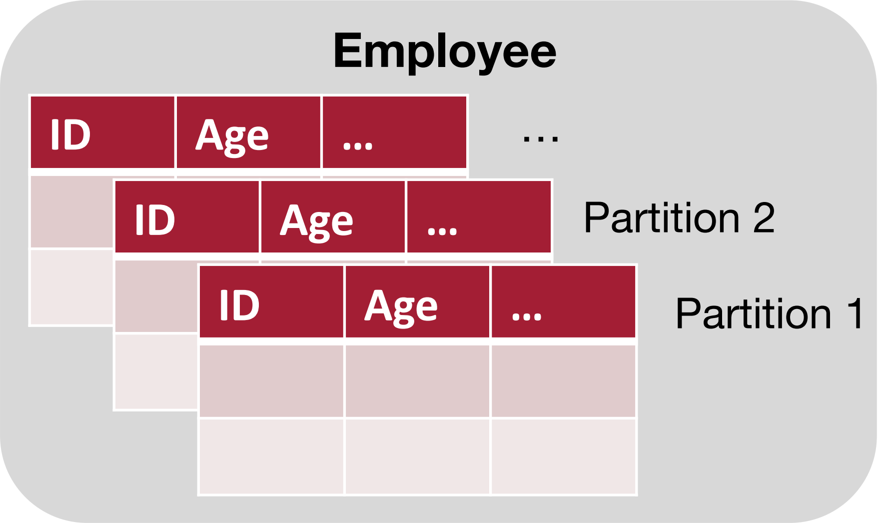
- Cache table data rather than query results for simplicity

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

FlexPushdownDB (FPDB) Overview

Design choices

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

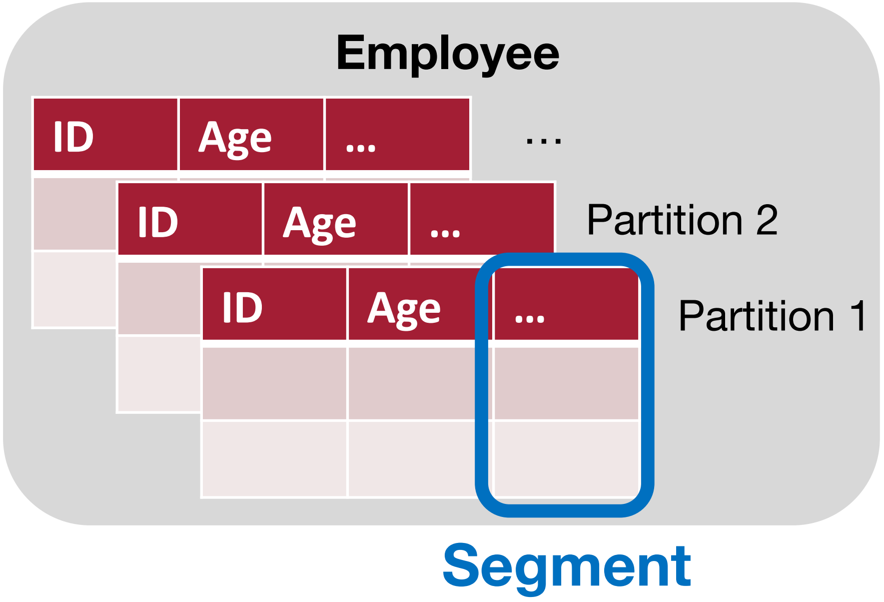


Source code: https://github.com/cloud-olap/FlexPushdownDB

FlexPushdownDB (FPDB) Overview

Design choices

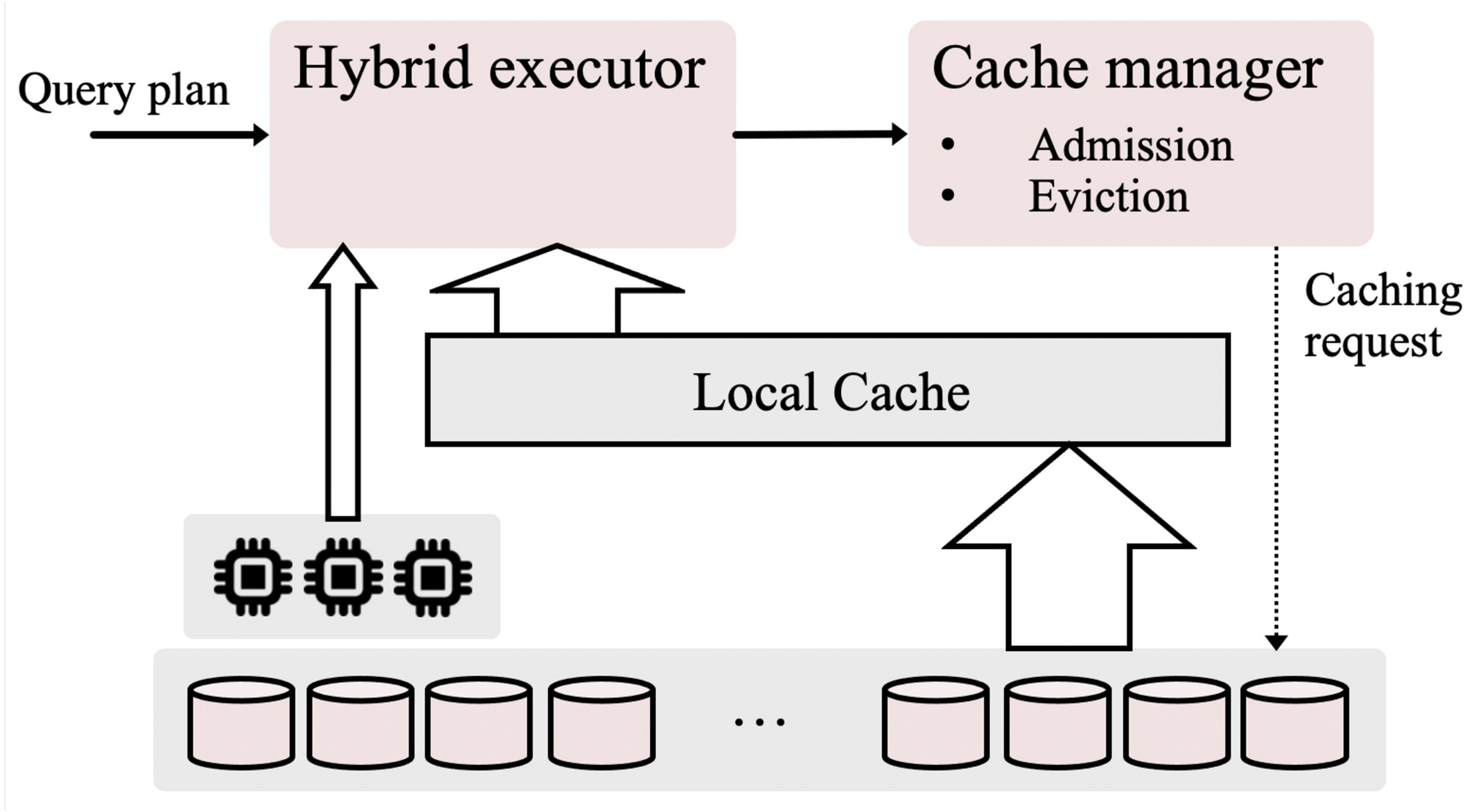
- Cache table data rather than query results for simplicity
- Segment as the caching granularity



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

FlexPushdownDB (FPDB) Overview

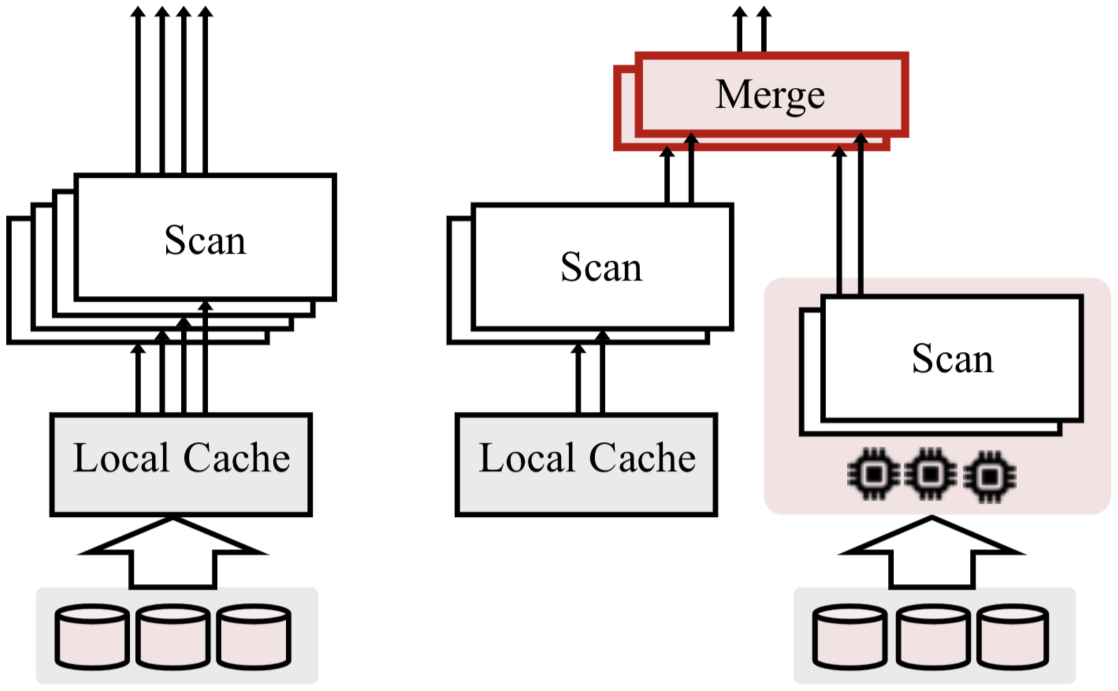
Main modules



FlexPushdownDB (FPDB)

Separable operators

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



(a) Original Query Plan (b) Separable Query Plan

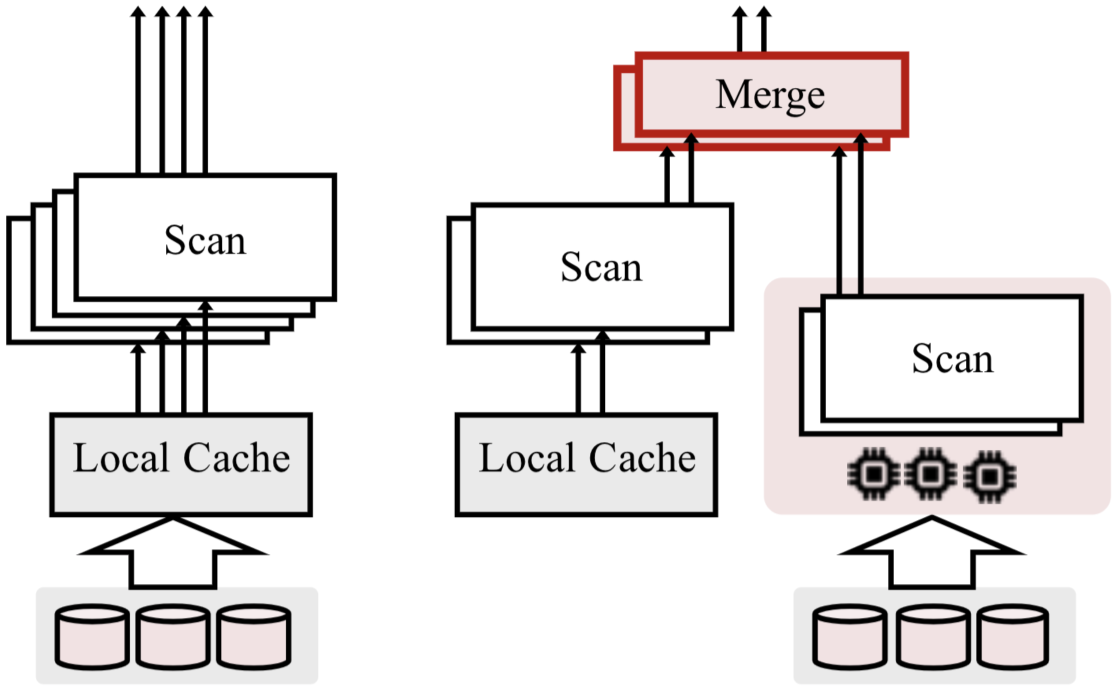
FlexPushdownDB (FPDB)

Separable operators

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

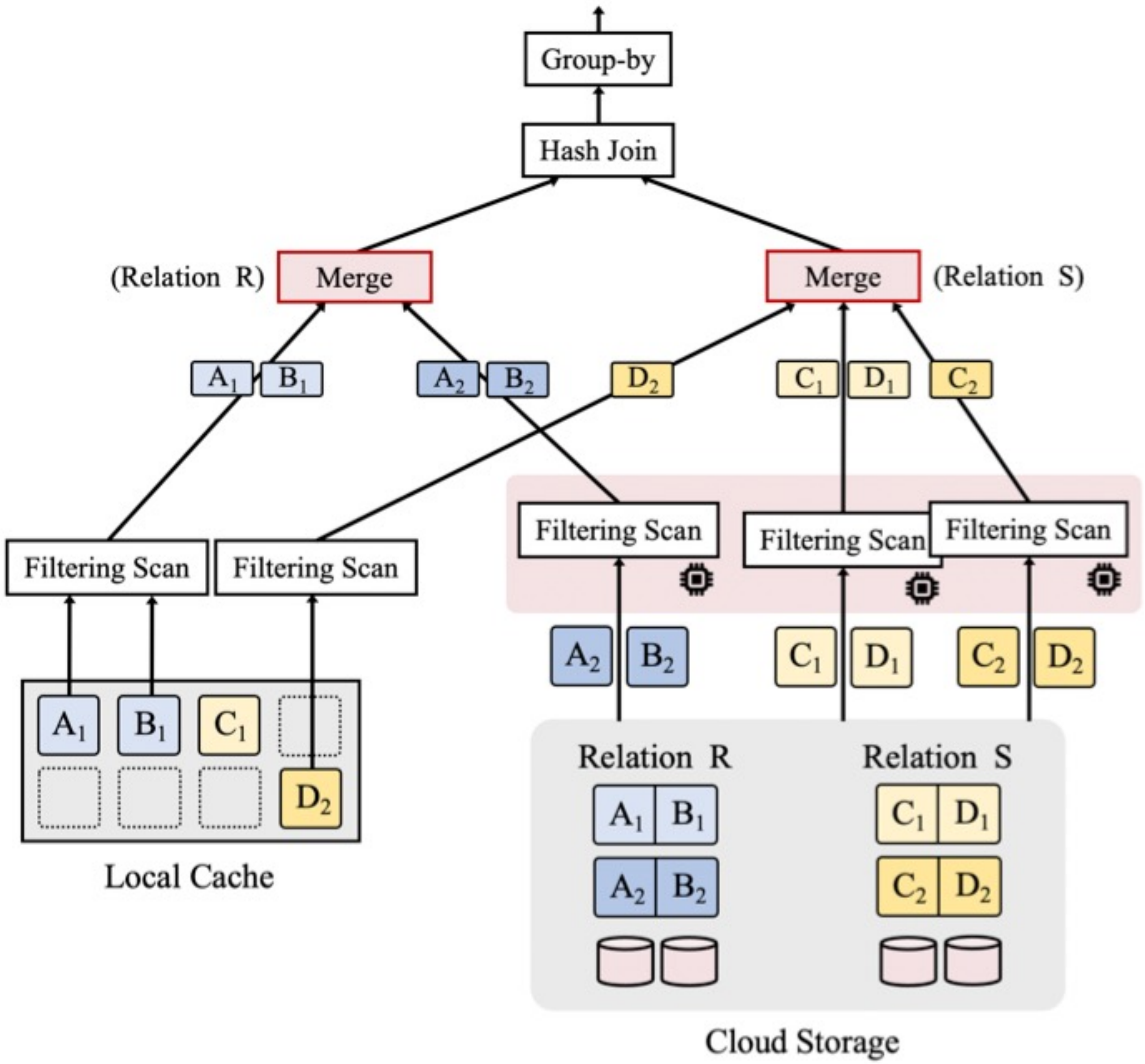
Query execution

- Heuristic: exploit caching when possible, otherwise pushdown as much as possible



(a) Original Query Plan (b) Separable Query Plan

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

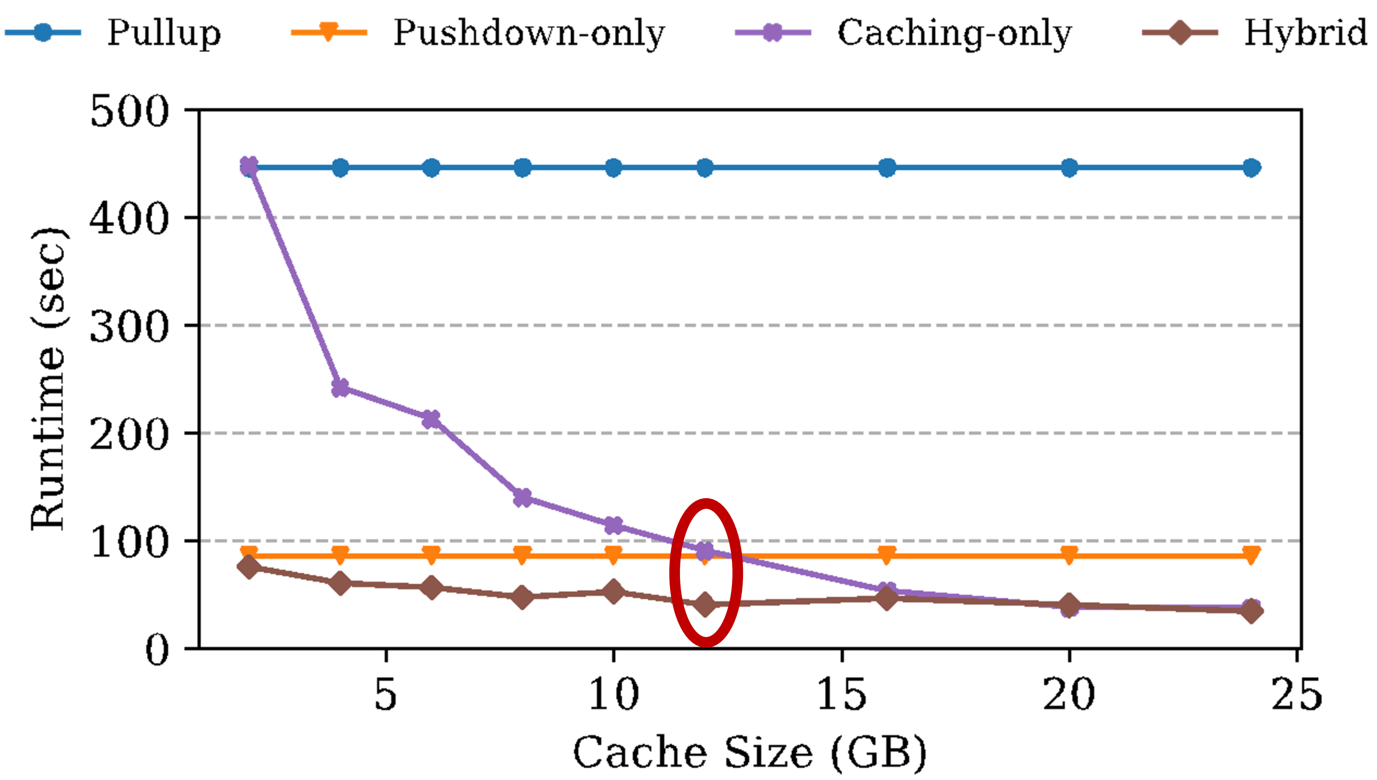
Traditional caching assumption: **Equal-size cache misses incur the same cost**

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

Weighted-LFU cache replacement policy

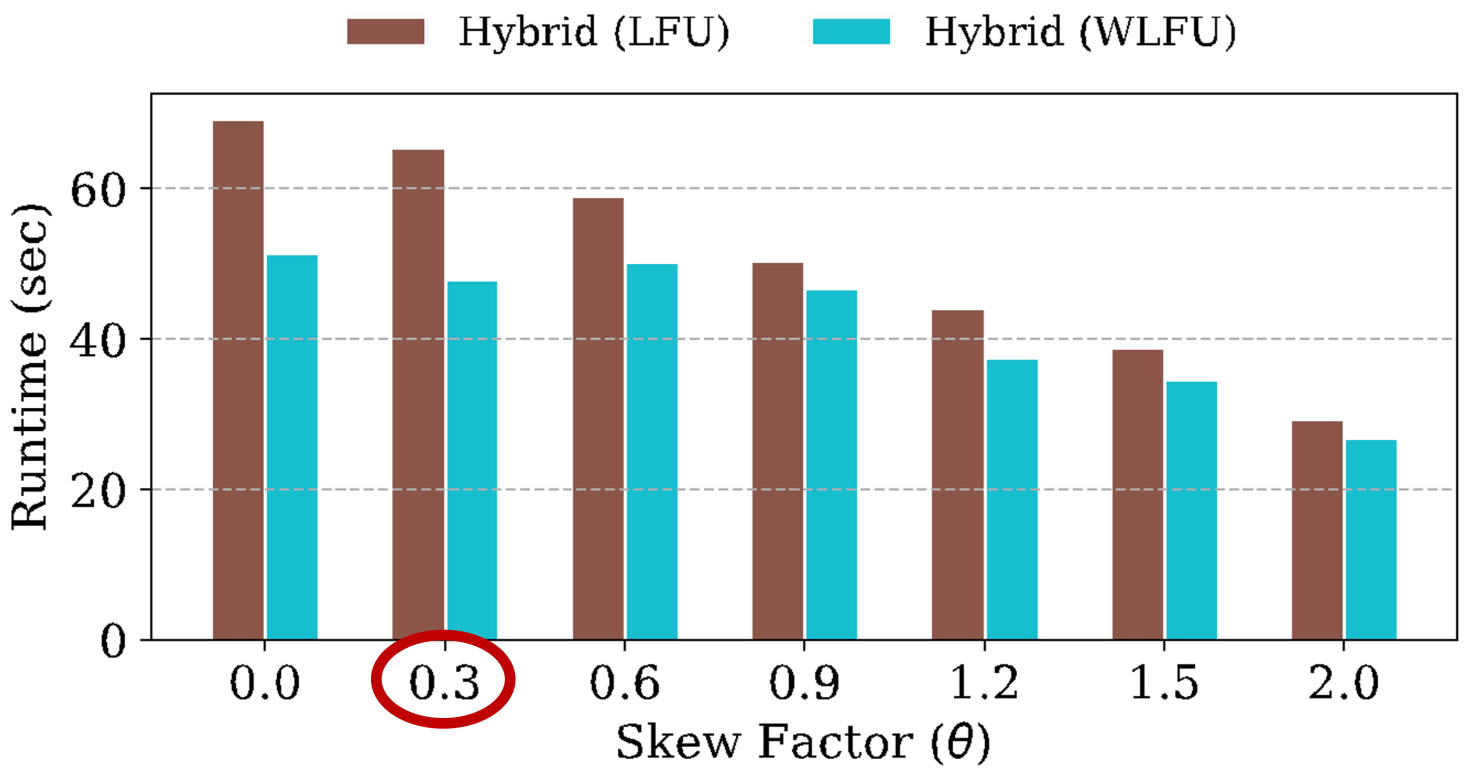
- Increment the frequency counter with the estimate miss cost
- Estimated miss cost = network cost + scan cost + compute cost

Performance Evaluation



Conclusion: FPDB outperforms baselines by 2.2x

Evaluation – Weighted-LFU



Weighted-LFU outperforms the baseline LFU by 37%

Evaluation – Resource Usage

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

Architecture	Pullup	PD-only	CA-only	Hybrid
Usage	460.9	37.1	112.6	7.9

Evaluation – Resource Usage

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

Architecture	Pullup	PD-only	CA-only	Hybrid
Usage	460.9	37.1	112.6	7.9

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

Architecture	Pullup	PD-only	CA-only	Hybrid
Compute	249.6	48.5	70.3	23.2
Storage	0.0	31.1	0.0	7.4
Total	249.6	79.6	70.3	30.6

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