Lecture #02

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Modern OLAP Databases

Andy Pavlo // 15-721 // Spring 2023

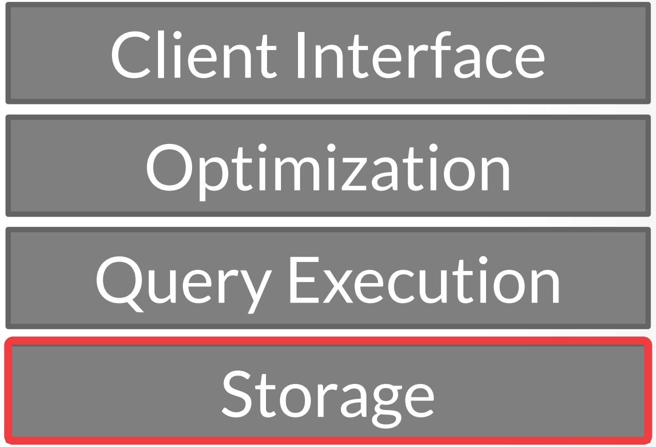
COURSE OUTLINE

Storage
- → Columnar Storage
- → Compression
- → Indexes

Query Execution:
- → Processing Models
- → Scheduling
- → Vectorization
- → Compilation
- → Joins
- → Materialized Views

Query Optimization

Network Interfaces



TODAY'S AGENDA

Query Execution

Distributed System Architectures

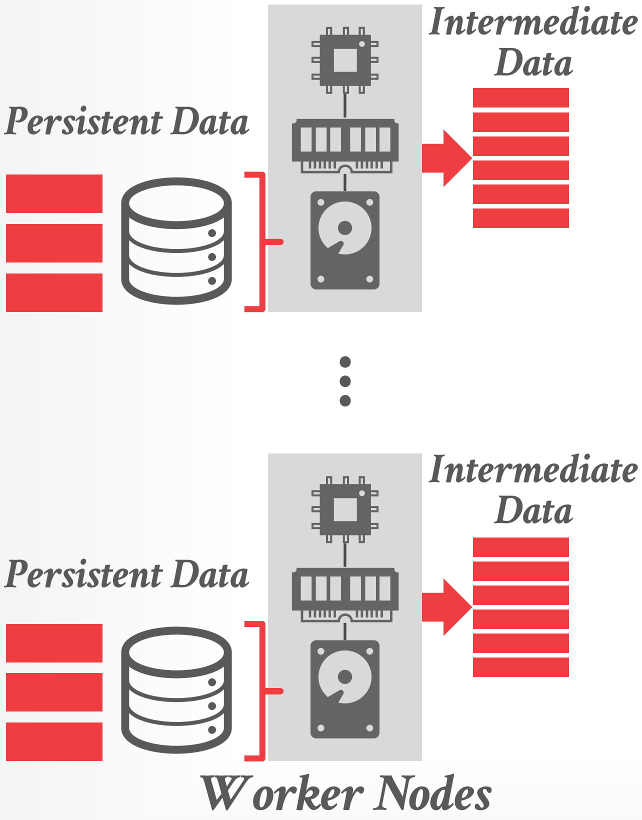
OLAP Commoditization

DISTRIBUTED QUERY EXECUTION

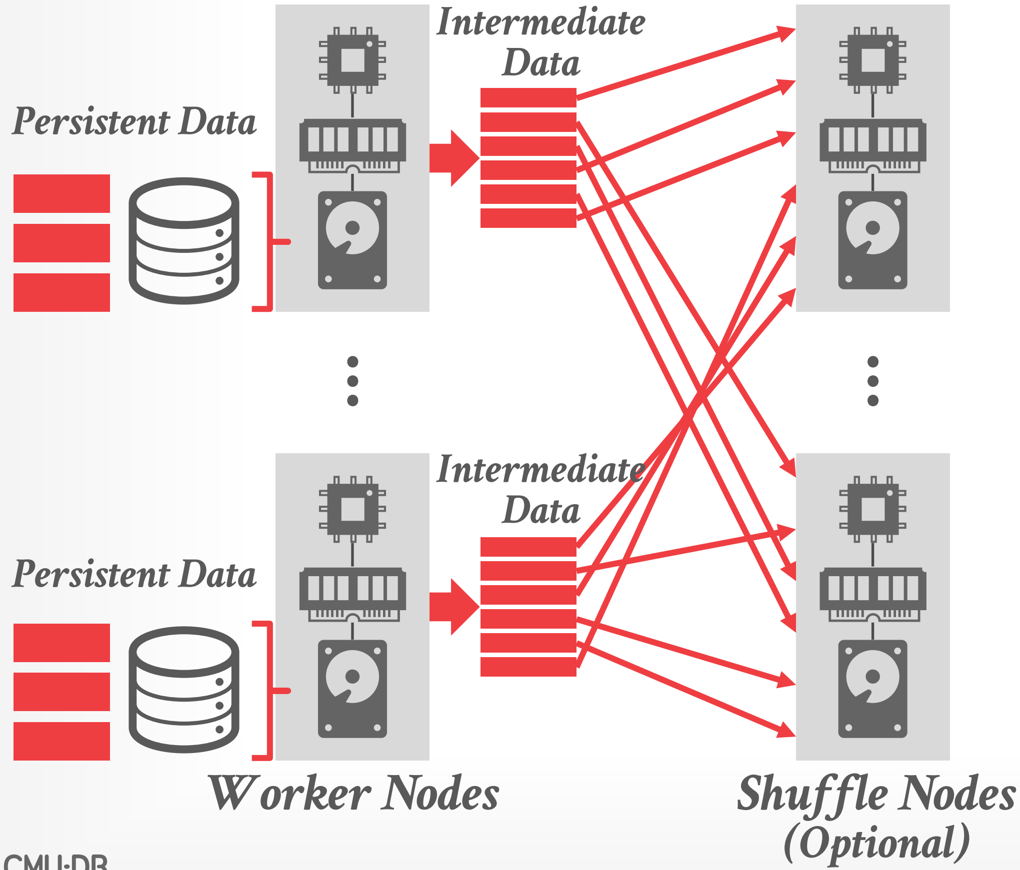
Executing an OLAP query in a distributed DBMS is roughly the same as on a single-node DBMS.
→ Query plan is a DAG of physical operators.

For each operator, the DBMS considers where input is coming from and where to send output.
→ Table Scans
→ Joins
→ Aggregations
→ Sorting

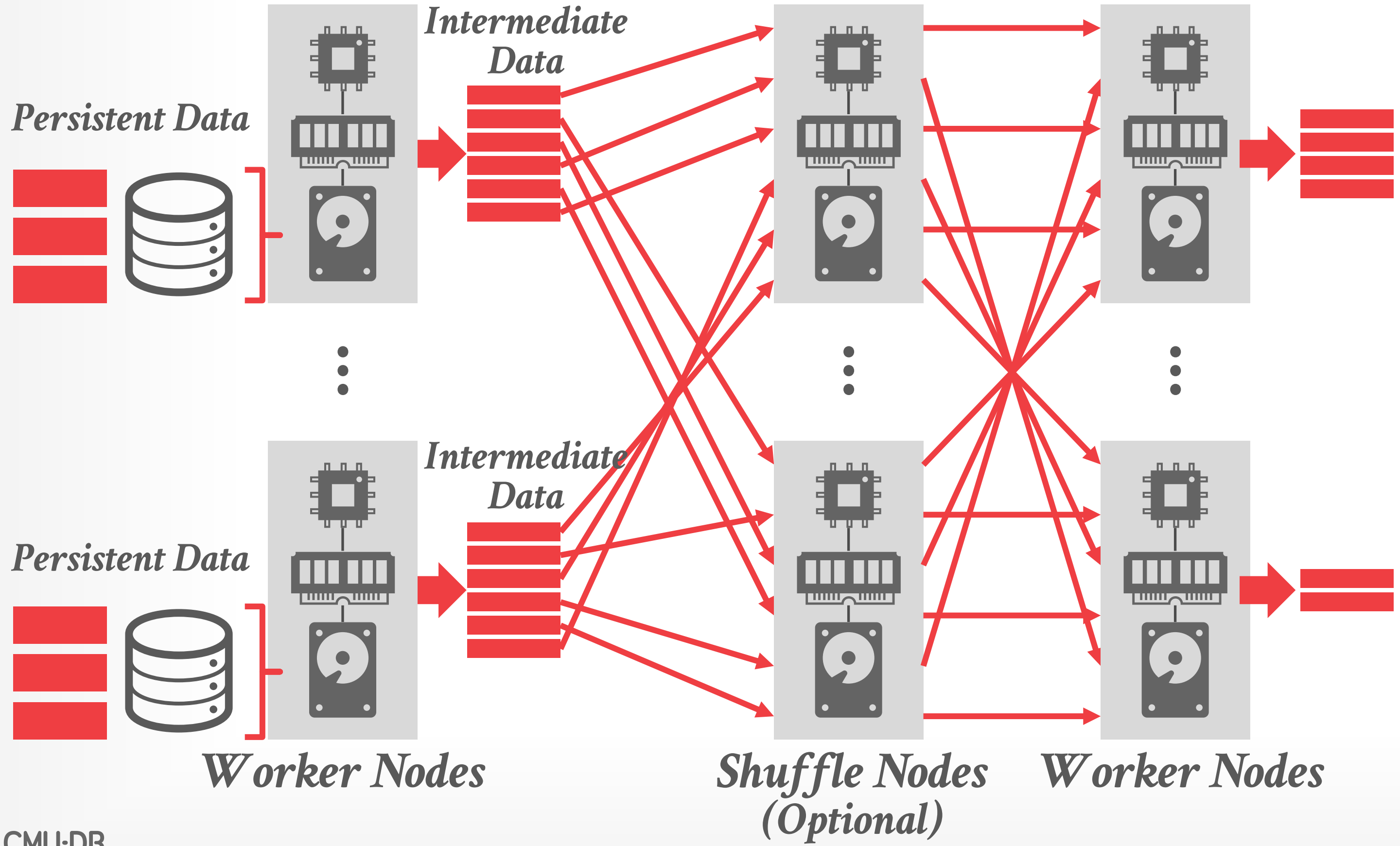
DISTRIBUTED QUERY EXECUTION



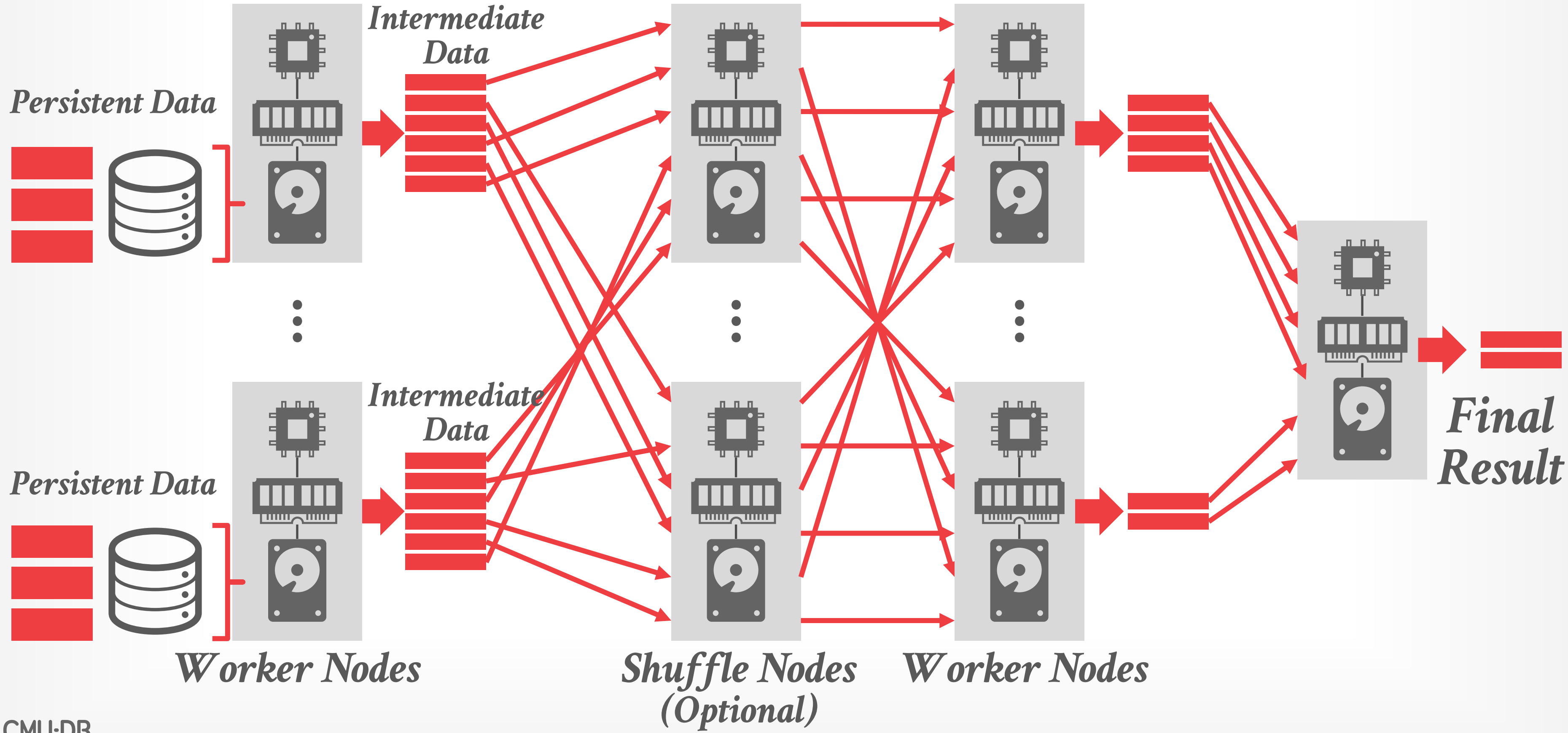
DISTRIBUTED QUERY EXECUTION



DISTRIBUTED QUERY EXECUTION



DISTRIBUTED QUERY EXECUTION



DATA CATEGORIES

Persistent Data:

→ The "source of record" for the database (e.g., tables).

→ Modern systems assume that these data files are immutable but can support updates by rewriting them.

Intermediate Data:

→ Short-lived artifacts produced by query operators during execution and then consumed by other operators.

→ The amount of intermediate data that a query generates has little to no correlation to amount of persistent data that it reads or the execution time.

BUILDING AN ELASTIC QUERY ENGINE ON DISAGGREGATED STORAGE
NSDI 2022

DISTRIBUTED SYSTEM ARCHITECTURE

A distributed DBMS's system architecture specifies the location of the database's persistent data files. This affects how nodes coordinate with each other and where they retrieve/store objects in the database.

Two approaches (not mutually exclusive):
→ **Push Query to Data**
→ **Pull Data to Query**

THE CASE FOR SHARED NOTHING
HPTS 1985

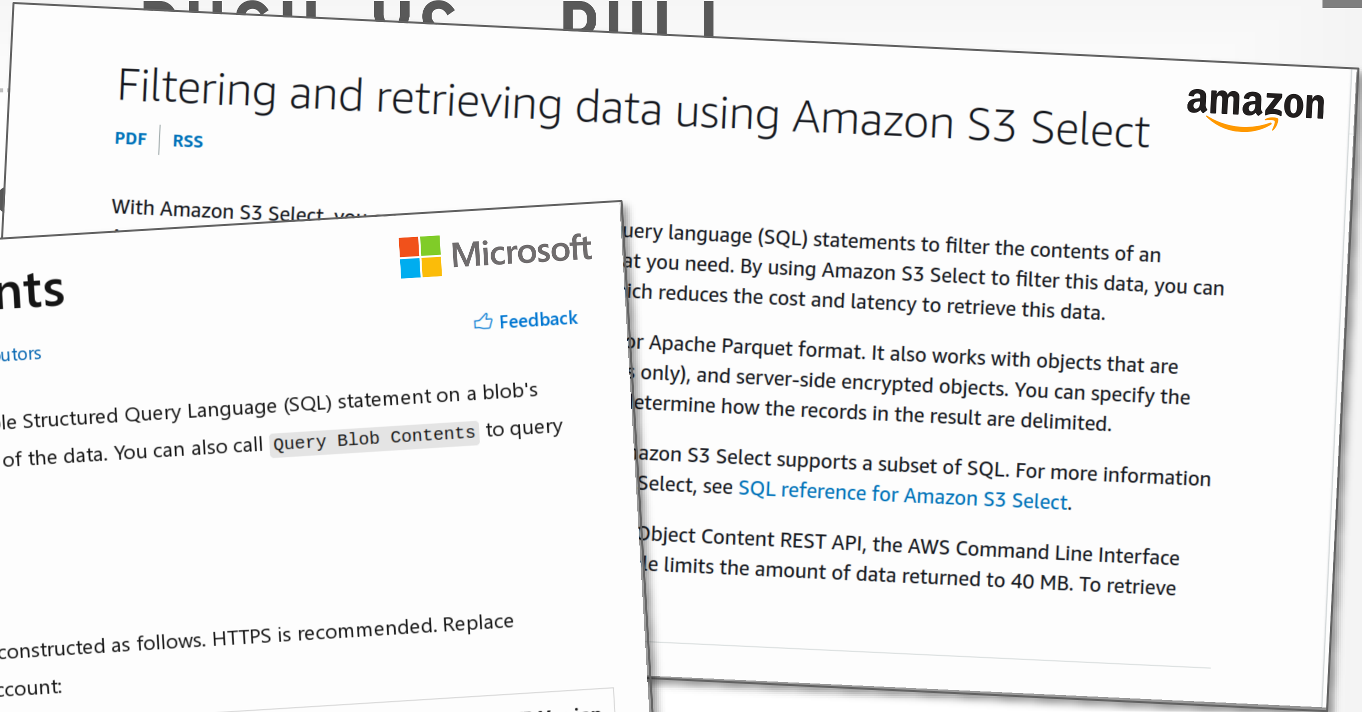
PUSH VS. PULL

Approach #1: Push Query to Data
→ Send the query (or a portion of it) to the node that contains the data.
→ Perform as much filtering and processing as possible where data resides before transmitting over network.

Approach #2: Pull Data to Query
→ Bring the data to the node that is executing a query that needs it for processing.
→ This is necessary when there is no compute resources available where persistent data files are located.

PUSH VS. PULL

Approa…



Filtering and retrieving data using Amazon S3 Select

amazon

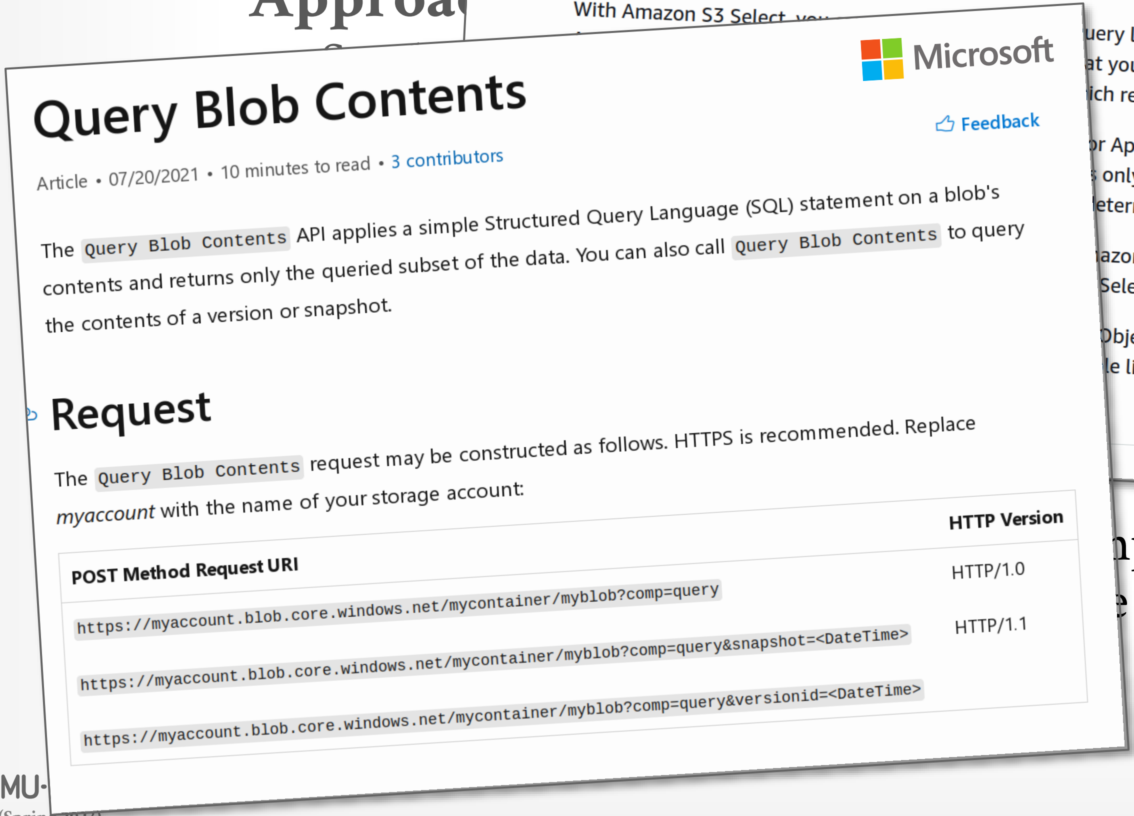
PDF | RSS

With Amazon S3 Select, you …uery language (SQL) statements to filter the contents of an …at you need. By using Amazon S3 Select to filter this data, you can …ich reduces the cost and latency to retrieve this data.

…or Apache Parquet format. It also works with objects that are …only), and server-side encrypted objects. You can specify the …etermine how the records in the result are delimited.

…azon S3 Select supports a subset of SQL. For more information …Select, see SQL reference for Amazon S3 Select.

…Object Content REST API, the AWS Command Line Interface …le limits the amount of data returned to 40 MB. To retrieve



Query Blob Contents

Microsoft

Feedback

Article • 07/20/2021 • 10 minutes to read • 3 contributors

The `Query Blob Contents` API applies a simple Structured Query Language (SQL) statement on a blob's contents and returns only the queried subset of the data. You can also call `Query Blob Contents` to query the contents of a version or snapshot.

Request

The `Query Blob Contents` request may be constructed as follows. HTTPS is recommended. Replace *myaccount* with the name of your storage account:

POST Method Request URI	HTTP Version
`https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=query`	HTTP/1.0
`https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=query&snapshot=<DateTime>`	HTTP/1.1
`https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=query&versionid=<DateTime>`	

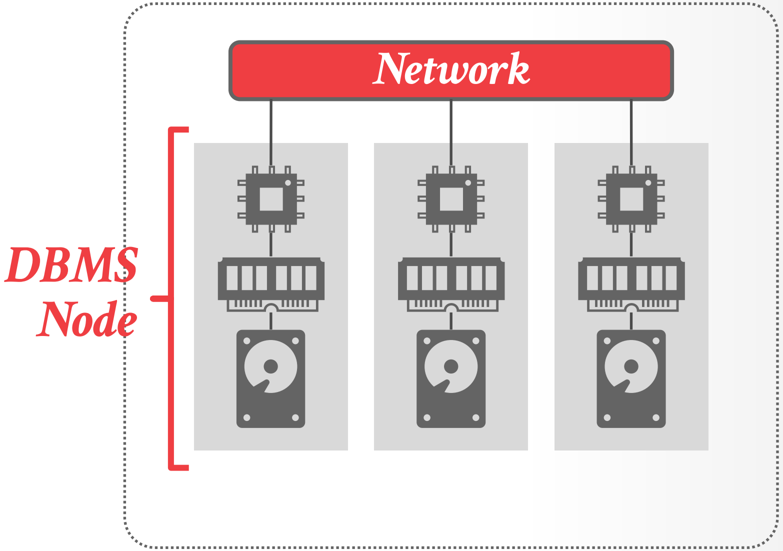
…mpute resources
…e located.

SHARED NOTHING

Each DBMS instance has its own CPU, memory, locally-attached disk.
→ Nodes only communicate with each other via network.

Database is partitioned into disjoint subsets across nodes.
→ Adding a new node requires physically moving data between nodes.

Since data is local, the DBMS can access it via POSIX API.

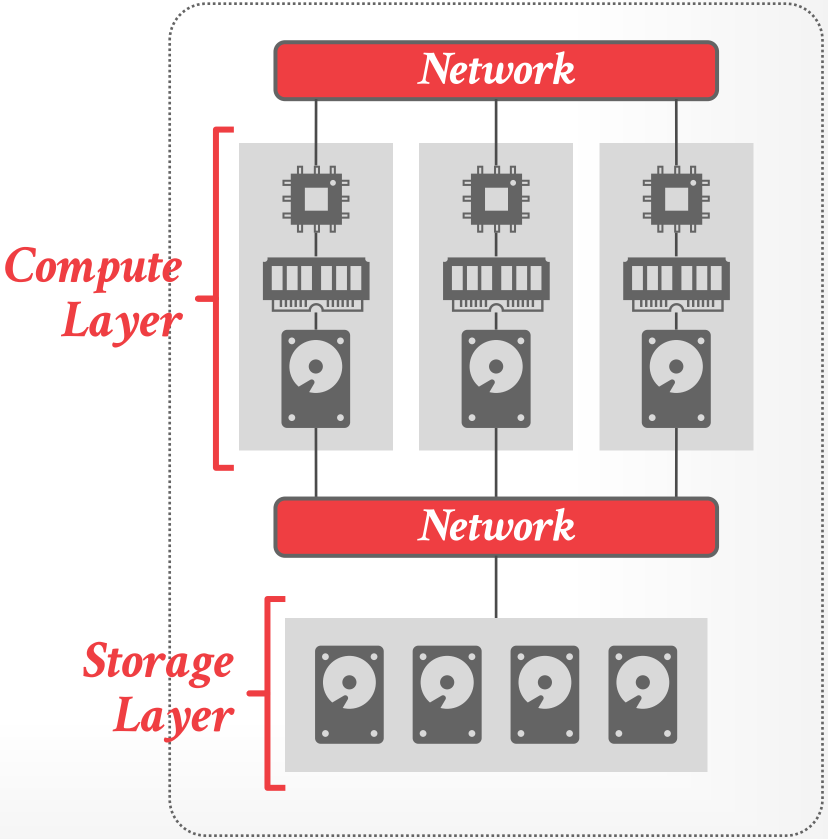


SHARED DISK

Each node accesses a single logical disk via an interconnect, but also have their own private memory and ephemeral storage.

→ Must send messages between nodes to learn about their current state.

Instead of a POSIX API, the DBMS accesses disk using a userspace API.



SYSTEM ARCHITECTURE

Choice #1: Shared Nothing:

→ Harder to scale capacity (data movement).
→ Potentially better performance & efficiency.
→ Apply filters where the data resides before transferring.

Choice #2: Shared Disk:

→ Scale compute layer independently from the storage layer.
→ Easy to shutdown idle compute layer resources.
→ May need to pull uncached persistent data from storage layer to compute layer before applying filters.

SHARED DISK

Traditionally the storage layer in shared-disk DBMSs were dedicated on-prem NAS.
→ Example: Oracle Exadata

Cloud **object stores** are now the prevailing storage target for modern OLAP DBMSs because they are "infinitely" scalable.
→ Examples: Amazon S3, Azure Blob, Google Cloud Storage

OBJECT STORES

Partition the database's tables (persistent data) into large, immutable files stored in an object store.

- → All attributes for a tuple are stored in the same file in a columnar layout (PAX).
- → Header (or footer) contains meta-data about columnar offsets, compression schemes, indexes, and zone maps.

The DBMS retrieves a block's header to determine what byte ranges it needs to retrieve (if any).

Each cloud vendor provides their own proprietary API to access data (**PUT**, **GET**, **DELETE**).

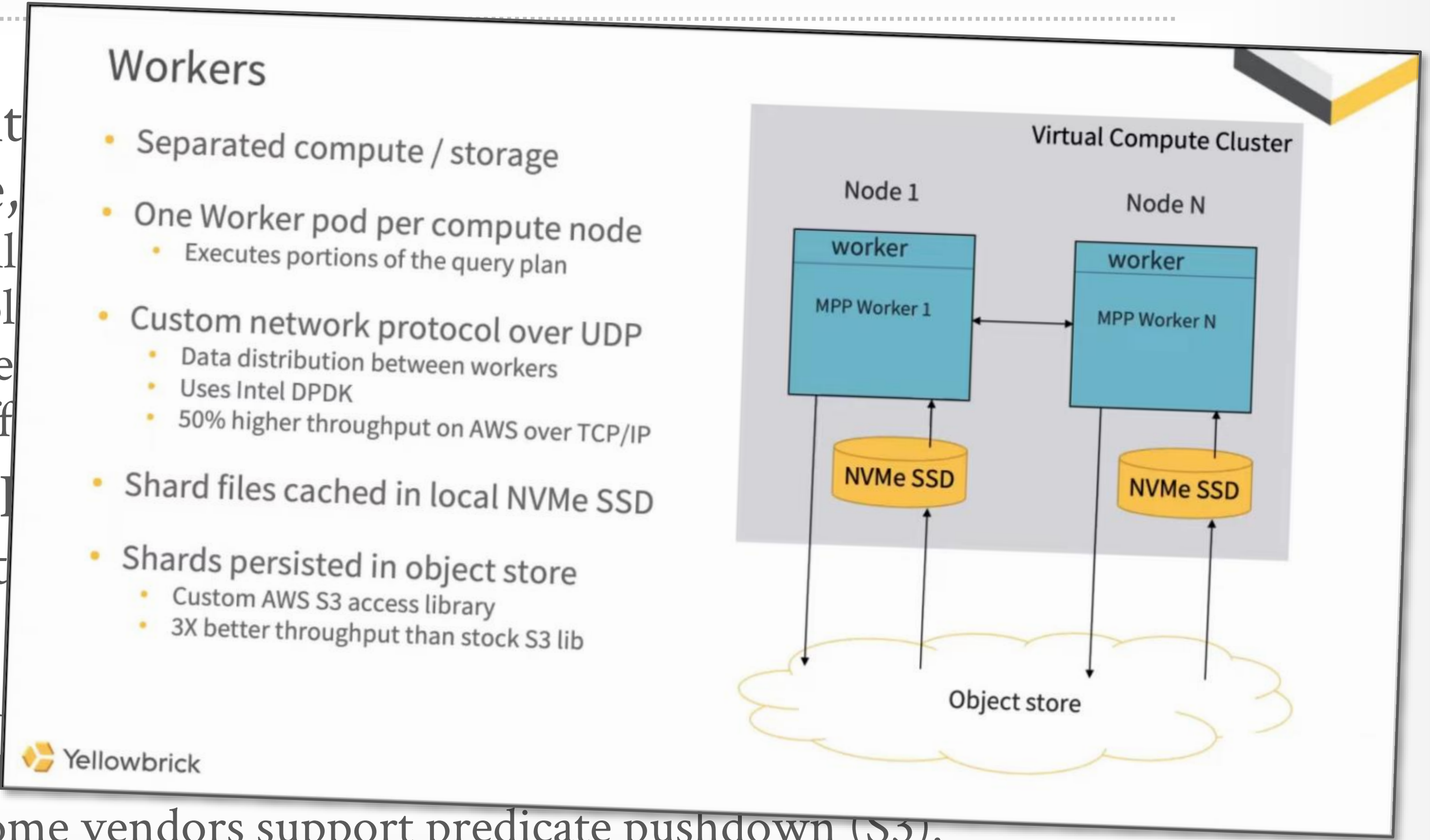
- → Some vendors support predicate pushdown (S3).

OBJECT STORES

Partit
large,
→ All
col
→ He
off

The I
what

Each
API t



→ Some vendors support predicate pushdown (S3).

ADDITIONAL TOPICS

File Formats

Table Partitioning

Data Ingestion / Updates / Discovery

Scheduling / Adaptivity

OBSERVATION

Snowflake is a monolithic system comprised of components built entirely in-house.

Most of the non-academic DBMSs we will cover this semester will have a similar overall architecture.

But this means that multiple organizations are writing the same DBMS software…

OLAP COMMODITIZATION

One recent trend of the last decade is the breakout OLAP engine sub-systems into standalone open-source components.

→ This is typically done by organizations <u>not</u> in the business of selling DBMS software.

Examples:

→ System Catalogs
→ Query Optimizers
→ File Format / Access Libraries
→ Execution Engines

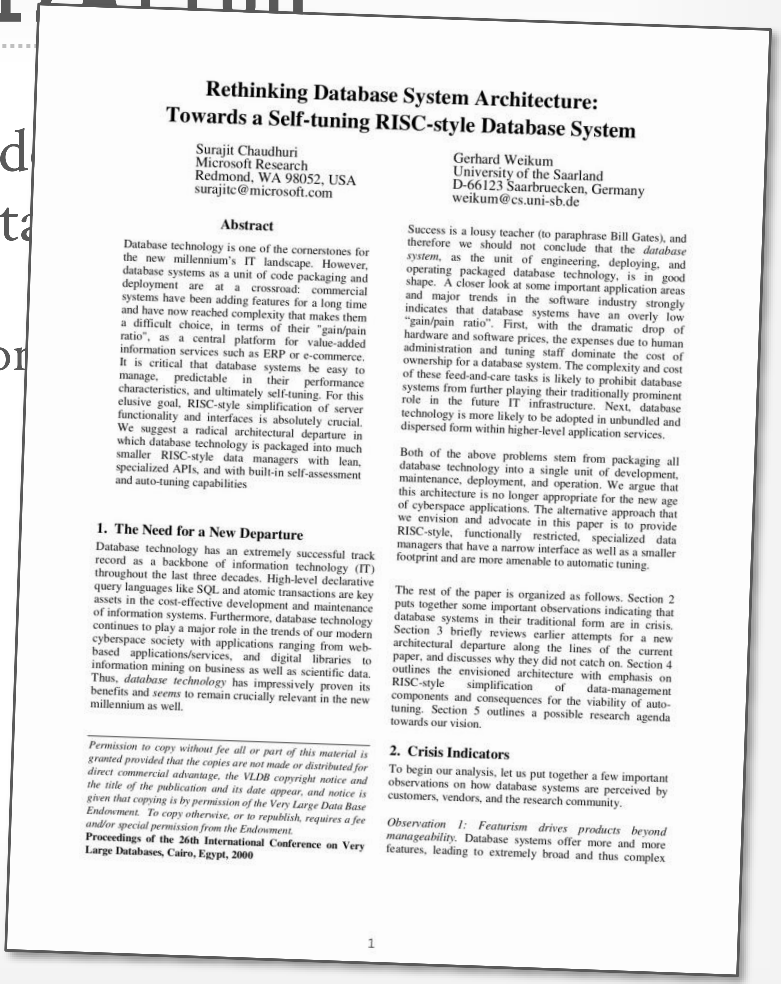
OLAP COMMODITIZATION

One recent trend of the last decade OLAP engine sub-systems into sta source components.

→ This is typically done by organization of selling DBMS software.

Examples:

→ System Catalogs
→ Query Optimizers
→ File Format / Access Libraries
→ Execution Engines



Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System

Surajit Chaudhuri
Microsoft Research
Redmond, WA 98052, USA
surajitc@microsoft.com

Gerhard Weikum
University of the Saarland
D-66123 Saarbruecken, Germany
weikum@cs.uni-sb.de

Abstract

Database technology is one of the cornerstones for the new millennium's IT landscape. However, database systems as a unit of code packaging and deployment are at a crossroad: commercial systems have been adding features for a long time and have now reached complexity that makes them a difficult choice, in terms of their "gain/pain ratio", as a central platform for value-added information services such as ERP or e-commerce. It is critical that database systems be easy to manage, predictable in their performance characteristics, and ultimately self-tuning. For this elusive goal, RISC-style simplification of server functionality and interfaces is absolutely crucial. We suggest a radical architectural departure in which database technology is packaged into much smaller RISC-style data managers with lean, specialized APIs, and with built-in self-assessment and auto-tuning capabilities

1. The Need for a New Departure

Database technology has an extremely successful track record as a backbone of information technology (IT) throughout the last three decades. High-level declarative query languages like SQL and atomic transactions are key assets in the cost-effective development and maintenance of information systems. Furthermore, database technology continues to play a major role in the trends of our modern cyberspace society with applications ranging from web-based applications/services, and digital libraries to information mining on business as well as scientific data. Thus, *database technology* has impressively proven its benefits and *seems* to remain crucially relevant in the new millennium as well.

Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000

Success is a lousy teacher (to paraphrase Bill Gates), and therefore we should not conclude that the *database system*, as the unit of engineering, deploying, and operating packaged database technology, is in good shape. A closer look at some important application areas and major trends in the software industry strongly indicates that database systems have an overly low "gain/pain ratio". First, with the dramatic drop of hardware and software prices, the expenses due to human administration and tuning staff dominate the cost of ownership for a database system. The complexity and cost of these feed-and-care tasks is likely to prohibit database systems from further playing their traditionally prominent role in the future IT infrastructure. Next, database technology is more likely to be adopted in unbundled and dispersed form within higher-level application services.

Both of the above problems stem from packaging all database technology into a single unit of development, maintenance, deployment, and operation. We argue that this architecture is no longer appropriate for the new age of cyberspace applications. The alternative approach that we envision and advocate in this paper is to provide RISC-style, functionally restricted, specialized data managers that have a narrow interface as well as a smaller footprint and are more amenable to automatic tuning.

The rest of the paper is organized as follows. Section 2 puts together some important observations indicating that database systems in their traditional form are in crisis. Section 3 briefly reviews earlier attempts for a new architectural departure along the lines of the current paper, and discusses why they did not catch on. Section 4 outlines the envisioned architecture with emphasis on RISC-style simplification of data-management components and consequences for the viability of auto-tuning. Section 5 outlines a possible research agenda towards our vision.

2. Crisis Indicators

To begin our analysis, let us put together a few important observations on how database systems are perceived by customers, vendors, and the research community.

Observation 1: Featurism drives products beyond manageability. Database systems offer more and more features, leading to extremely broad and thus complex

SYSTEM CATALOGS

A DBMS tracks a database's schema (table, columns) and data files in its catalog.

→ If the DBMS is on the data ingestion path, then it can maintain the catalog incrementally.

→ If an external process adds data files, then it also needs to update the catalog so that the DBMS is aware of them.

Notable implementations:

→ HCatalog

→ Google Data Catalog

→ Amazon Glue Data Catalog

QUERY OPTIMIZERS

Extendible search engine framework for heuristic- and cost-based query optimization.

→ DBMS provides transformation rules and cost estimates.
→ Framework returns either a logical or physical query plan.

This is the hardest part to build in any DBMS.

Notable implementations:

→ Greenplum Orca
→ Apache Calcite

ORCA: A MODULAR QUERY OPTIMIZER ARCHITECTURE FOR BIG DATA
SIGMOD 2014

APACHE CALCITE: A FOUNDATIONAL FRAMEWORK FOR OPTIMIZED QUERY PROCESSING OVER HETEROGENEOUS DATA SOURCES
SIGMOD 2018

FILE FORMATS

Most DBMSs use a proprietary on-disk binary file format for their databases. The only way to share data between systems is to convert data into a common text-based format

→ Examples: CSV, JSON, XML

There are open-source binary file formats that make it easier to access data across systems and libraries for extracting data from files.

→ Libraries provide an iterator interface to retrieve (batched) columns from files.

UNIVERSAL FORMATS

Apache Parquet (2013)
→ Compressed columnar storage from Cloudera/Twitter

Apache ORC (2013)
→ Compressed columnar storage from Apache Hive.

Apache CarbonData (2013)
→ Compressed columnar storage with indexes from Huawei.

Apache Iceberg (2017)
→ Flexible data format that supports schema evolution from Netflix.

HDF5 (1998)
→ Multi-dimensional arrays for scientific workloads.

Apache Arrow (2016)
→ In-memory compressed columnar storage from Pandas/Dremio.

EXECUTION ENGINES

Standalone libraries for executing vectorized query operators on columnar data.

→ Input is a DAG of physical operators.
→ Require external scheduling and orchestration.

Notable implementations:

→ Velox
→ DataFusion
→ Intel OAP

VELOX: META'S UNIFIED EXECUTION ENGINE
VLDB 2022

CONCLUSION

Today was about understanding the high-level context of what modern OLAP DBMSs look like.

→ Fundamentally these new DBMSs are not different than previous distributed/parallel DBMSs except for the prevalence of a cloud-based object store for shared disk.

Our focus for the rest of the semester will be about state-of-the-art implementations of these systems' components.

NEXT CLASS

Storage Models

Data Representation

Partitioning

Catalogs