Data management in the era of bigdata and machine learning

Farouk Toumani

LIMOS, CNRS, Clermont Auvergne INP, UCA

Summer School
Modern Machine Learning: Foundations and Applications
School of Information and Communication Technology
Hanoi University of Science and Technology
September 11^{th}, 2023

Modern advanced analytics

$\Rightarrow$ Software systems for advanced analytics over large and complex datasets are becoming critical for digital applications

- Machine Learning (ML) has become quite popular
- On track to impact many industries
- Needs to process large amounts of data
- Arbitrary complex processing scripts

Modern advanced analytics

⇒ Software systems for advanced analytics over large and complex datasets are becoming critical for digital applications

- Machine Learning (ML) has become quite popular
- On track to impact many industries
- Needs to process large amounts of data
- Arbitrary complex processing scripts

⇒ Several ML frameworks emerged in the recent years and are being widely adopted

- Support (advanced) data analytics: statistical analysis, data mining, deep learning (DL), ...
- Equiped with high-level APIs to express computations over (large) datasets
- Execution engine to run analytical operations efficiently

Modern advanced analytics

⇒ Software systems for advanced analytics over large and complex datasets are becoming critical for digital applications

- Machine Learning (ML) has become quite popular
- On track to impact many industries
- Needs to process large amounts of data
- Arbitrary complex processing scripts

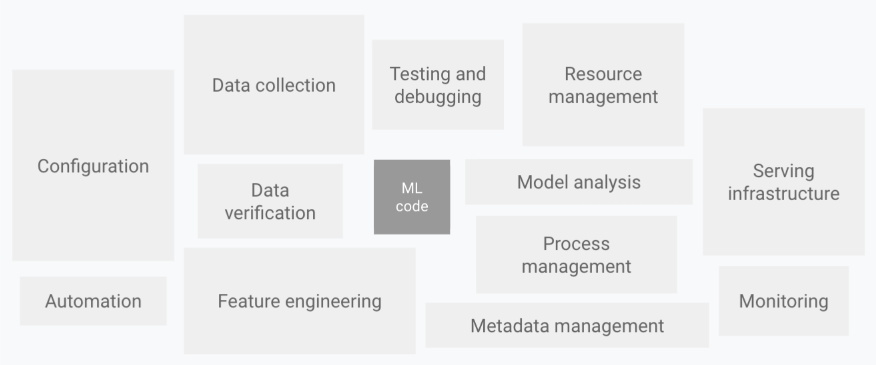
⇒ Several ML frameworks emerged in the recent years and are being widely adopted

- Support (advanced) data analytics: statistical analysis, data mining, deep learning (DL), . . .
- Equiped with high-level APIs to express computations over (large) datasets
- Execution engine to run analytical operations efficiently

⇒ High-value data in the enterprise is typically stored in databases, data warehouses, or data lakes

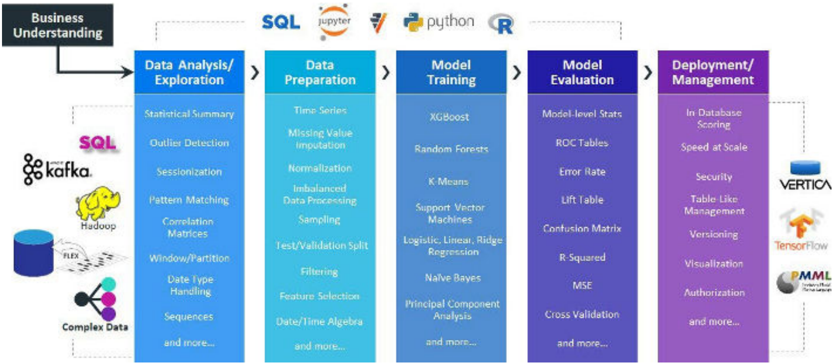
Machine learning pipeline

Several pre-processing steps (from Google)



Machine learning pipeline

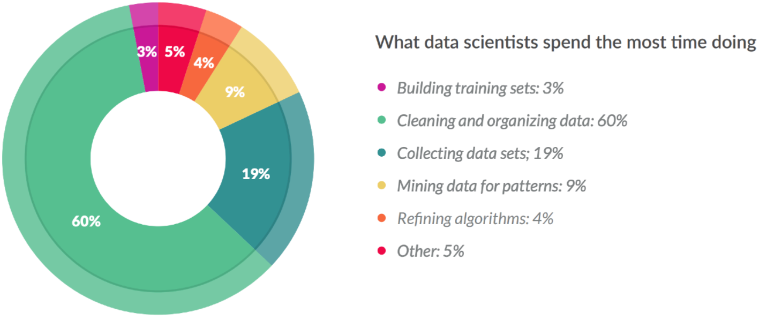
From Vetica



Machine learning pipeline

- A cumberscome pre-processing process
    - Transfer cost: move the (complete) data from the database server to the client machine
    - Limited client memory: data might exceed typical client memory amounts
    - Memory management: e.g., Pandas operations often create copies of the internal data and therefore occupy more of the client's RAM
- 80% of ML users time/effort (often more) spent on data issues!

Machine learning pipeline



https://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf

The rise of Artificial Intelligence

- **Data-centric AI**
  - Model-centric lifecycle
    - Primarily focus on identifying more effective models to improve AI performance while keeping the data largely unchanged
    - Overlooks the potential quality issues and undesirable flaws of data (missing values, incorrect labels, and anomalies, . . .)
  - Data-centric lifecycle
    - Systematic engineering of data to build AI systems

    $\Rightarrow$ Shifting the focus from model to data

    $\Rightarrow$ Need of a robust and scalable data management system
- **Declarative AI**
  - Next wave of ML systems: allow a *larger amount of people, potentially without coding skills, to perform ML tasks* [MR21a]

What can data management do for machine learning?

Outline

What is data management?

In the early days, data-centric applications were built directly on top of file systems, which leads to:

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation
- Integrity problems

$\Rightarrow$ Database and Database Managament Systems (DBMSs)

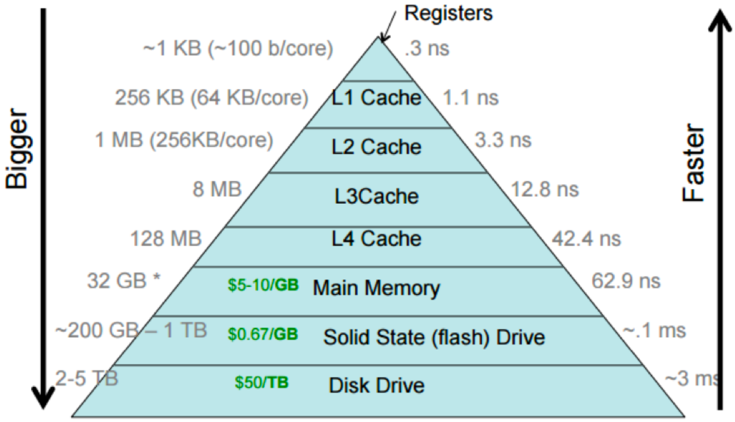
$\Rightarrow$ Core component of most (modern) computer applications

$\Rightarrow$ Data-centeric AI

Some foundational principles

- Model to abstract how data is represented and manipulated
  - $\Rightarrow$ Logical and physical data independence
- Declarative query language
- Automatic optimization at different levels
  - Architecture of the system: disk-based vs. memory-based, centralized vs. distributed vs. parallel, ...
  - Query processing
- Concurrency control
- Automatic failure recovery

Storage hierarchy



Source: https://cs.brown.edu/courses/csci1310/2020/assign/labs/lab4.html

DBMS's design goals

- Manage data that exceed the amount of memory available

⇒ **Disk-based architecture**: reduce the number of I/O operations

- Temporal control
  - When the data gets read or written to the disk?

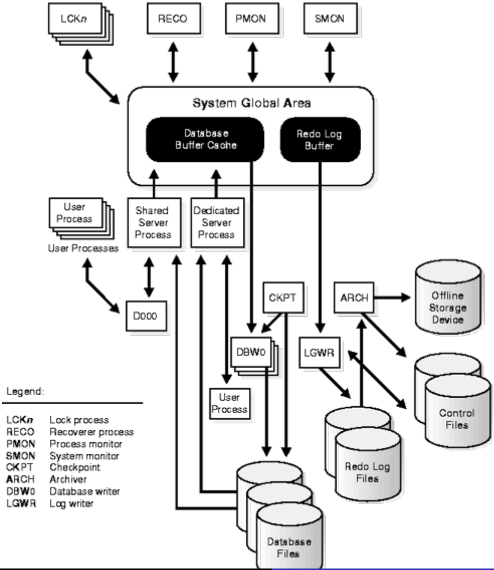
  ⇒ using buffering techniques
- Spatial control
  - Where to store the data on the disk?

  ⇒ Using the DBMS's physical model

Buffering

- Use a buffer cache to reduce the number of I/O operations
  - Reading from a buffer cache instead of physically reading from the disk
    - ⇒ Takes benefit from concurrent accesses to shared data
    - ⇒ Read-ahead (early reads, speculative reads): detailed future access pattern knowledge is available to the DBMS
  - Writing in the buffer cache
    - ⇒ Lazy writes (write-behind, delayed, batched writes)
- A logic to control when to write blocks to disk to ensure the correctness of the database

DBMS Architecture



Spatial control

- Access patterns
  - Random access
  - Sequential access

$\Rightarrow$ Sequential bandwidth to and from disk is between 10 and 100 times faster than random access, and this ratio is increasing

$\Rightarrow$ Physical model to control the spatial locality

- Logical storage structures
  - Tablespaces
  - Database blocks
  - Extents
  - Segments : Table, Index, cluster, ...
- Physical storage structures: files

Index

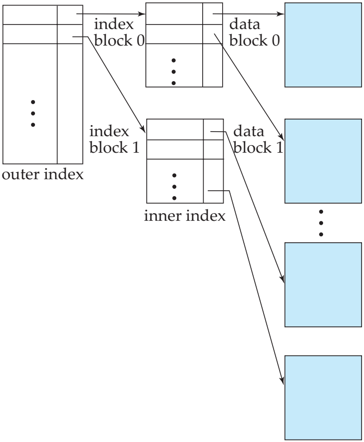
Example:Sparse index

10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

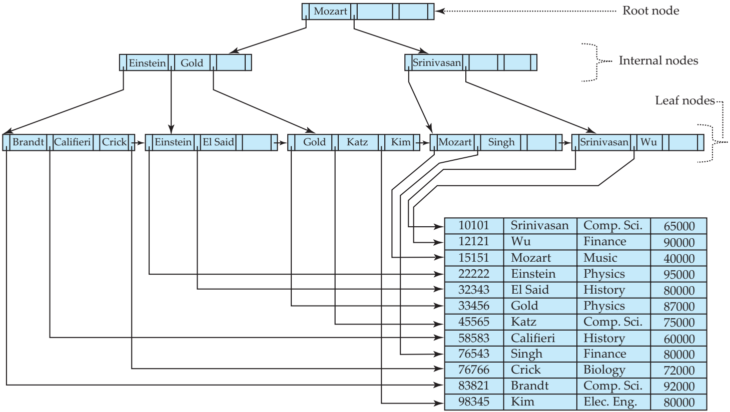
Index

Example: Two level sparse index



Index

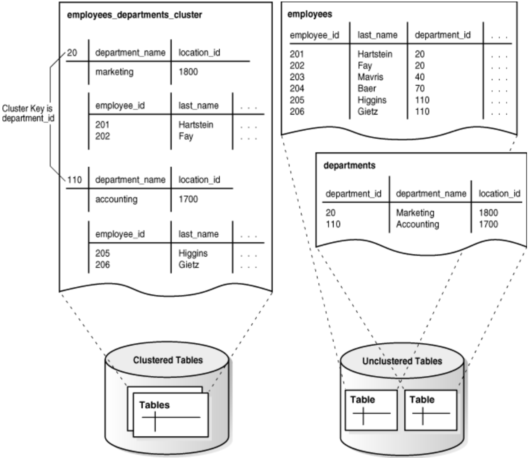
Example:B+-Tree index



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Clustered tables

Example



Source: `https://docs.oracle.com/en/database/oracle/oracle-database/19/`

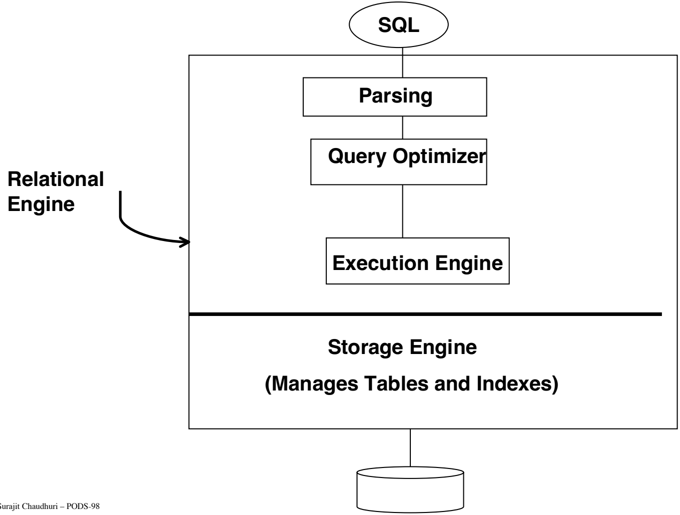
Query processing

- SQL is a declarative language
  - $\Rightarrow$ Physical data Independence
  - $\Rightarrow$ Needs to be compiled into a program

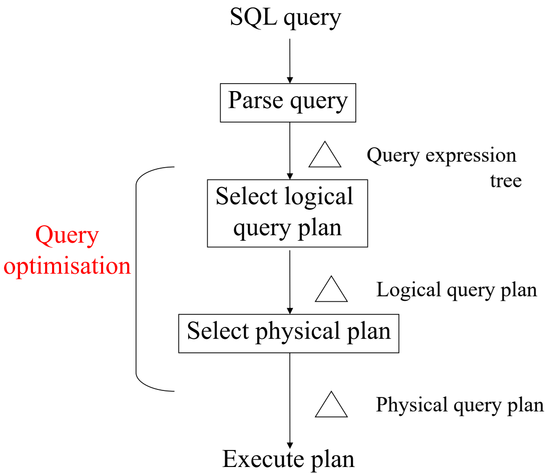
$\Rightarrow$ Opens rooms for optimization

- Compiles the query into a program that consumes the least ressources

Query processing



Query processing



Query plan

- Logical query plan
  - Relational algebra: $\sigma, \pi, \bowtie, \cup, \cap, \ldots$
- Physical query plan
  - Physical operators
    - Sequential scan
    - Index scan
    - Filter
    - Join: Nested-Loop, Sort-Merge, Indexed Nested-loop...

$\Rightarrow$ Operator trees

Logical query optimization

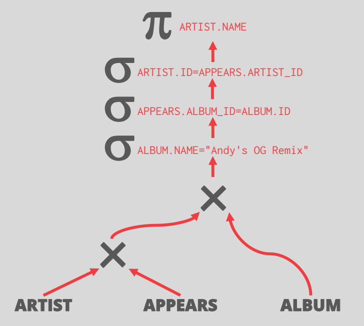
Example

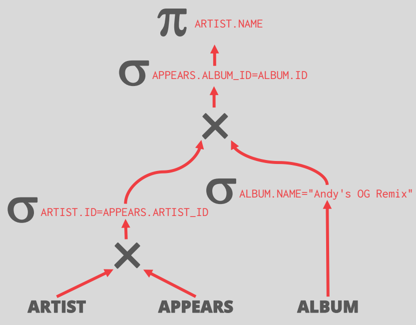
```
Select A.NAME
   From ARTIST, APPEARS, ALBUM
   Where ARTIST.ID=APPEARS.ARTIST_ID AND
   APPEARS.ALBUM_ID=ALBUM.ID AND ALBUM.NAME="Andy's OG Remix"
```

Query optimization

Example

- **Rule-based** optimization
- Rules are based on the algebraic properties of the operators
- Strategy of the application of the rules based on heuristics

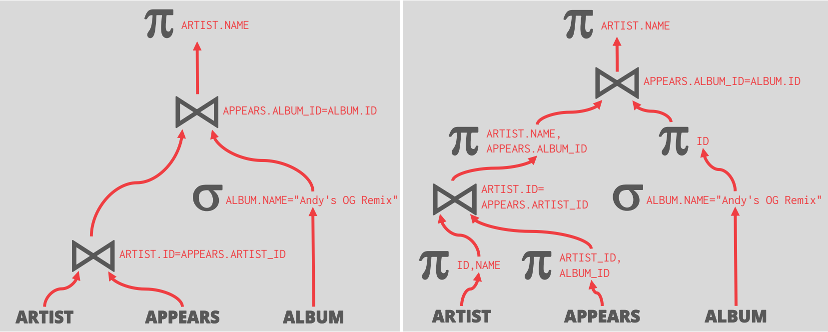




Source: https://15445.courses.cs.cmu.edu/fall2022/slides/14-optimization.pdf

Query optimization

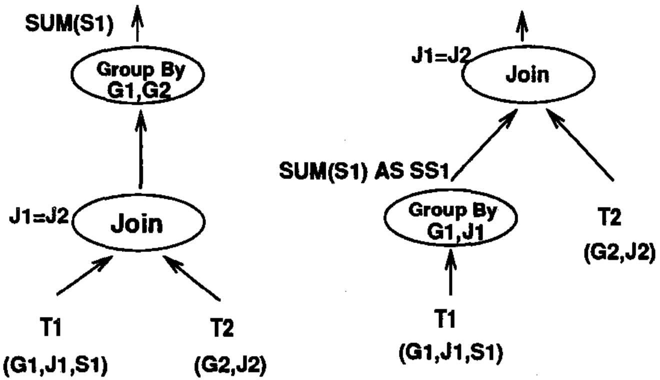
Example



Source: https://15445.courses.cs.cmu.edu/fall2022/slides/14-optimization.pdf

Query optimization

Example - Eager/Lazy aggregation [YL95]



(a) Group-by Pull up

(b) Group-by Push down

Query optimization

Subqueries and aggregation[GLJ01]

- Query: *"finds customers who have ordered more than $1,000,000"*
- Database

  `Customer(c_custkey, ...)`

  `Orders(o_ordkey, ..., o_custkey)`

Query optimization

Subqueries and aggregation[GLJ01]

- Query: *"finds customers who have ordered more than $1,000,000"*
- Database

  `Customer(c_custkey, ...)`

  `Orders(o_ordkey, ..., o_custkey)`
- Query

```
Select c_custkey
    From Customer
    Where 1000000 <
         (Select sum(o_totalprice)
         From Orders
         Where o_custkey = c_custkey)
```

Query optimization

Subqueries and aggregation[GLJ01]

- Query: *"finds customers who have ordered more than $1,000,000"*
- Database
  `Customer(c_custkey, ...)`
  `Orders(o_ordkey, ..., o_custkey)`
- Query

```
Select c_custkey
    From Customer
    Where 1000000 <
         (Select sum(o_totalprice)
         From Orders
         Where o_custkey = c_custkey)
```

⇒ **Correlated execution**: the subquery is executed as many times as there are employees

Query optimization

Subqueries and aggregation[GLJ01]

- Outerjoin, then aggregate

```
Select c custkey
    from customer left outer join
    orders on o custkey = c custkey group by c custkey
    having 1000000 < sum(o totalprice)
```

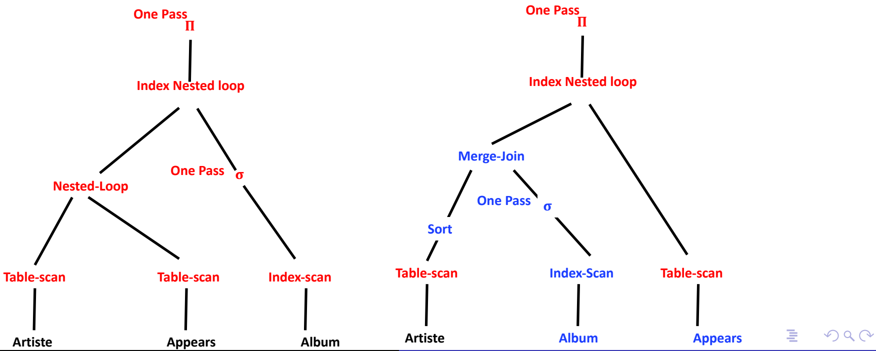
- Aggregate, then join

```
select c custkey from customer,
    (select o custkey from orders group by c custkey
    having 1000000 < sum(o totalprice)) as AggResult
    where o custkey = c custkey
```

Physical plans

Example

- **Cost-based** optimization
  - Cardinality estimation for SQL expressions
  - Cost estimation for SQL execution plans (or partial plans)
  - A dynamic programming based algorithm to search the space of execution plans



User Defined Functions (UDFs)

- **Complex processing tasks** cannot be expressed in SQL

$\Rightarrow$ UDF: procedural extension of SQL

- Support of various programming languages: PL/SQL, Transact-SQL, Java, C#, **Python, R**, ...
- Widely used in practical applications: e.g., more than 10M of T-SQL UDFs in use in the Microsoft Azure SQL database service [RPE$^+$17]

User Defined Functions (UDFs)

Example

```
Create function service_level(int ckey) returns char(10) as
Begin
     float totalbusiness; string level;
     Select sum(totalprice) into :totalbusiness
     From orders Where custkey=:ckey;
     if(totalbusiness > 1000000) level = "Platinum";
     else if(totalbusiness > 500000) level = "Gold";
     else level = "Regular";
     return level;
End
```

User Defined Functions (UDFs)

Example

```
Create function service_level(int ckey) returns char(10) as
Begin
     float totalbusiness; string level;
     Select sum(totalprice) into :totalbusiness
     From orders Where custkey=:ckey;
     if(totalbusiness > 1000000) level = "Platinum";
     else if(totalbusiness > 500000) level = "Gold";
     else level = "Regular";
     return level;
End
```

Query: `Select custkey, service_level(custkey) From customer`

User Defined Functions (UDFs)

Pros and cons

(+) Achieves modularity and code reuse across SQL queries

(+) Enable to express complex business rules (and recently ML algorithms)

(+) Support various programming languages

(-) **Performance overhead** due to the impedance mismatch between two paradigms: declarative paradigm of SQL and imperative paradigm of procedural code

- Naive execution strategies: context switching, data copies, data conversion, materialization of intermediate results, . . .
- Limited query optimization
  - Semantics of UDF is not known to the optimizer
  - Cost-based optimization

Optimization of queries with UDFs

Example of decorrelation of UDF Invocations [SRC+14]

```
With e as (Select custkey, sum(totalprice) as totalbusiness
From orders Group by custkey);

Select c.custkey,
Case
     When e.totalbusiness > 1000000 Then "Platinum"
     When e.totalbusiness > 500000 Then "Gold"
Else "Regular"
End as service
From customer c left outer join e on c.custkey=e.custkey;
```

Optimization of queries with UDFs

Example of decorrelation of UDF Invocations [SRC+14]

```
With e as (Select custkey, sum(totalprice) as totalbusiness
From orders Group by custkey);
Select c.custkey,
Case
     When e.totalbusiness > 1000000 Then "Platinum"
     When e.totalbusiness > 500000 Then "Gold"
Else "Regular"
End as service
From customer c left outer join e on c.custkey=e.custkey;
```

- Decorrelated query: more efficient execution plan
- Set-oriented execution plan: expands the space of alternative plans for the optimizer
- Decorrelating UDF invocations is a complex task due to the presence of various imperative constructs

User Defined Aggregation Functions (UDAFs)

- Built-in aggregate functions: initially **min, max, sum, count, avg** and now many other functions
- UDF (c.f., pros and cons)
- New mechanism: User Defined Aggregation function (UDAF)
  - **Init**: initialization
  - **Accumulate**: usually computes partial aggregation (intermediate result)
  - **Merge**: merges two intermediate result (Optionnel)
  - **Terminate**: computes the final result

User Defined Aggregation Functions (UDAFs)

Example

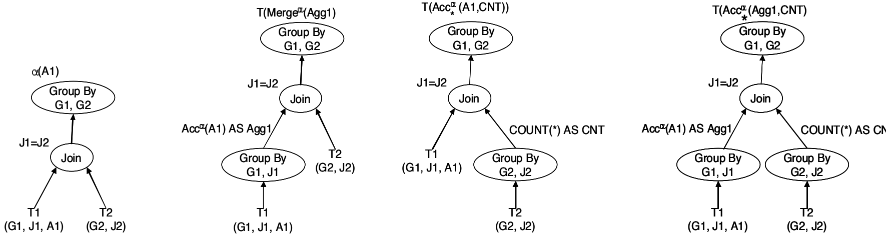
- The product function

```
public class Prod {
     double temp;
     public void Init() { temp = 1; }
     public void Accumulate(double newVal) {
          temp = temp * newVal; }
     public double Terminate() { return temp; }
     public void Merge(Prod other) {
          temp = temp * other.temp; }
}
```

Optimization of queries with UDAFs

Examples from [Coh06]

Select $T_1.G_1, T_2.G_2, \alpha(A1)$ **From** T_1, T_2
Where $T_1.J_1 = T_2.J_2$ **Group By** G_1, G_2



Initial Plan **Eager Group-BY** **Eager Count** **Double Eager**

Big Data era

It is all about **V**s ...

- **V**olume
- **V**ariety
- **V**elocity
- **V**eracity
- ...

Why did data become big?

Modern society generates a huge amount of data

Why did data become big?

Modern society generates a huge amount of data

- **Technological (r)evolution**: Storage (SSD, ..), networks and telecommunications (Wifi, 3G, CPL etc.), Calculation processors, Graphics cards, sensors, cameras, miniaturization, reduction of energy consumption / price
- **Data acquisition**: massive and in real time, Web, Invisible computing, laptops, web, IoT, CPS (Cyber Physical Systems)
- **Connected world** Networks (Cluster) of machines, Sensor networks, Mobile networks, Social networks, Internet of things
- **Software/architecture developments**: Virtualization, services Computing grids, Cloud computing Software: visualization, data analysis, simulation, learning, ..

Data management in the era of big data

- A huge amount of data that **cannot be stored and processed by traditional database** solutions
- Revitalization of research and development in data management

  New classes of data management systems

  - NoSQL wave
  - NewSQL data management systems

NoSQL data management systems

- Different data models
  - Key-Value
  - Document
  - Wide-column
  - Graph
  - ...
- Main techniques
  - Sharding
  - Replication
  - Large shared nothing clusters
  - Limited queries capabilities

$\Rightarrow$ Horizontal scalability

$\Rightarrow$ Tolerance to failures

$\Rightarrow$ Trade-off Consistency, Availability, Partition-tolerance (CAP)

CAP theorem [GL02]

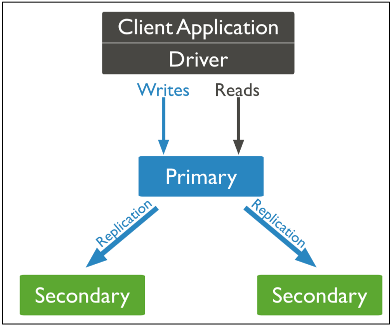
- **C**onsistency: every read receives the most recent write or errors out
- **A**vailability: every request receives a response
- **P**artition tolerance: tolerance of a storage system to failure of a network partition (system continues to operate even if some of the messages are dropped/delayed)
- CAP theorem
  - **AP:** Available and Partition Tolerant
  - **CP:** Consistent and Partition Tolerant
  - **CA:** Not Partition Tolerant

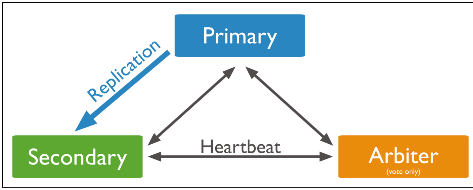
Data replication

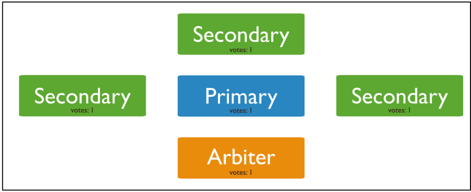
- Replication: multiple copies of the **same** data set on different database servers
- Main goals
  - High availability
  - Fault tolerance against the loss of a single database server
  - Increased read capacity
  - Increase data locality
  - Data copies for dedicated purposes (backup, disaster recovery, ...)

Replication in MongoDB

Replica set

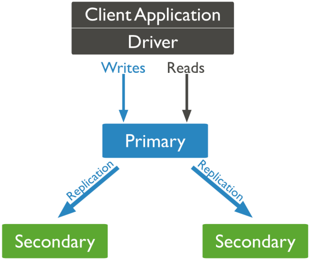


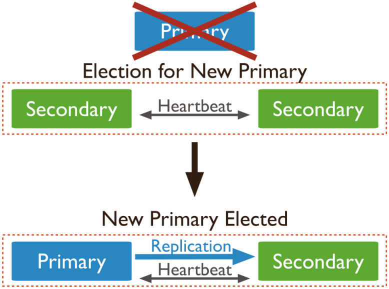




MongoDB Replication

Automatic failover





Sharding

Data partioning in the NoSQL era

- **Challenges:** Large data sets, high query rates, ...
  $\Rightarrow$ Horizontal scaling v.s. Vertical scaling
- **Sharding:** a method for distributing data across multiple machines
  - Data distribution that is nearly transparent to the application
  - Support deployments with very large data sets and high throughput operations

  $\Rightarrow$ Document-based databases: sharding data at the collection level

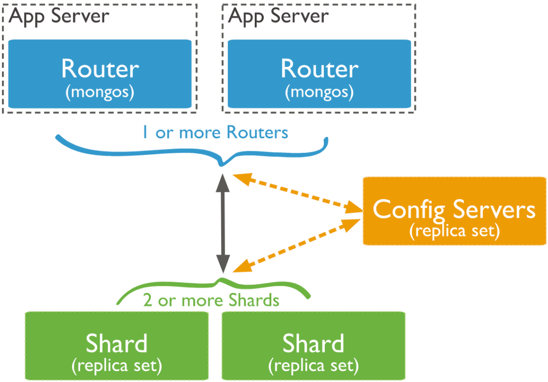
  $\Rightarrow$ Horizontal scaling: dynamic sharding (add/remove a server)

Advantages of Sharding

- Distribution of read and write operations
- Horizontal scalability
- Very suited to queries that include the shard key or the prefix of a compound shard key
- Storage Capacity
- High Availability

MongoDB

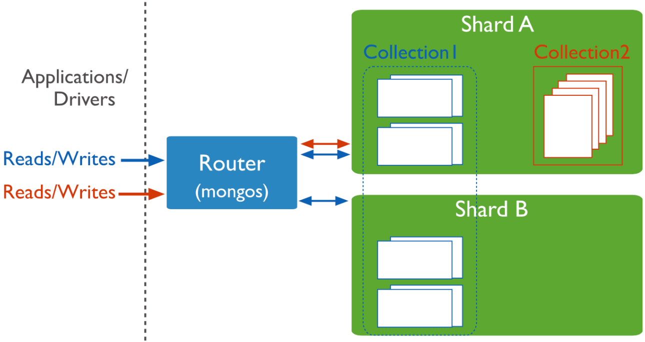
Sharding



https://docs.mongodb.com/manual/sharding/

Sharding

Routers and collections



https://docs.mongodb.com/manual/core/distributed-queries/

Targeted Operations vs. Broadcast Operations

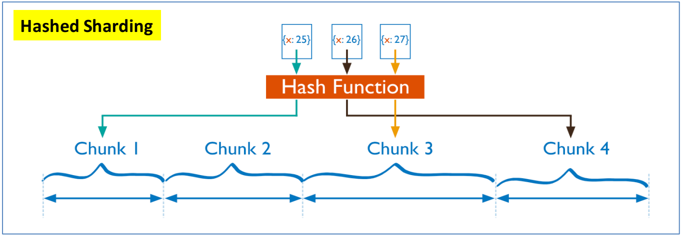
- Targeted operations
  - Queries that include the shard key or the prefix of a compound shard key
  - Queries routed to a specific shard or set of shards
- Broadcast Operations
  - Queries broadcasted to all shards
  - Responses from all shards are merged

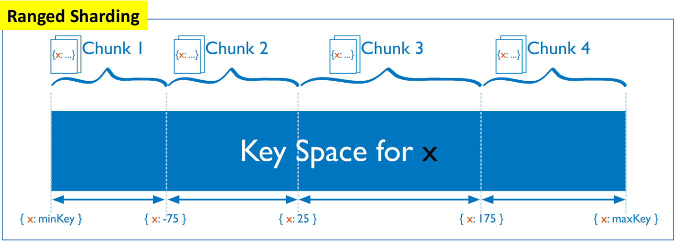
Sharding strategies

Two sharding strategies

- Hashed Sharding
  - Compute a hash of the shard key field's value
  - Each chunk is assigned a range based on the hashed shard key values
  - (+) Facilitates even data distribution, especially in data sets where the shard key **changes monotonically**
  - (-) Range-based queries on the shard key are less likely to target a single shard
- Range Sharding
  - Divide data into ranges based on the shard key values
  - Each chunk is assigned a range based on the shard key values
  - (+) Range-based queries on the shard key
  - (-) Poorly considered shard keys can result in uneven distribution of data

Sharding strategies





MapReduce paradigm

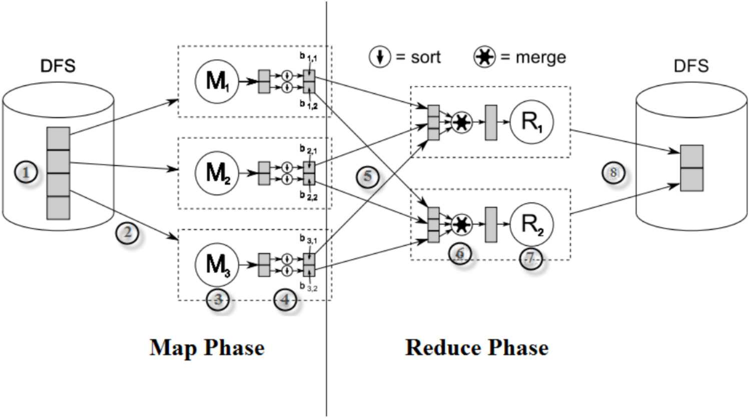
Motivation

- Massively parallel programs
  - Simple parallel programming model
  - Scalability
  - Fault tolerance
- Moving programs not data !!

MapReduce framework

- A process in two steps
  - A Map step : execution of a user provided map function
  - A Reduce step : execution of a user provided reduce function
- Main advantages/drawbacks
  - (+) Make parallelism transparent to the programmer
    - Muliple instances of the map function are executed in parallel
    - Muliple instances of the reduce function are executed in parallel
  - (+) Massively parallel model
    - Fault tolerance: failures have *local effects*
    - Horizontal scalability
  - (-) Suitable only for simple problems (highly parallelizable)
  - (-) Initialization cost

MapReduce framework



Design of MapReduce programs

- Map function
  - tuple at a time funcion
  - returns <key, value> pairs
- Reduce function
  - Takes as input a paire <key, list(values)>

MongoDB MapReduce

Example

{ _id: ObjectId("50a8240b927d5d8b5891743c"),
cust_id: "abc123",
ord_date: new Date("Oct 04, 2012"),
status: 'A',
price: 25,
items: [{ sku: "mmm", qty: 5, price: 2.5 },
{ sku: "nnn", qty: 5, price: 2.5 }] }

- Return the Total Price Per Customer
- Calculate Order and Total Quantity with Average Quantity Per Item

MongoDB MapReduce

Example

- Return the Total Price Per Customer

```
db.orders.mapReduce(
    function() {emit(this.cust_id, this.price);},
    function(keyCustId, valuesPrices)
        {return Array.sum(valuesPrices)},
    { out: "myResult" } )
```

MongoDB MapReduce

Exampe

- Calculate Order and Total Quantity with Average Quantity Per Item

```
var mapFunction2 = function() {
  for (var idx = 0; idx < this.items.length; idx++) {
      var key = this.items[idx].sku;
      var value = {
            count: 1,
            qty: this.items[idx].qty };
      emit(key, value); } };
```

MongoDB MapReduce

Example

```
var reduceFunction2 = function(keySKU, countObjVals) {
    reducedVal = { count: 0, qty: 0 };
    for (var idx = 0; idx < countObjVals.length; idx++) {
        reducedVal.count += countObjVals[idx].count;
        reducedVal.qty += countObjVals[idx].qty; }
    return reducedVal; };

var finalizeFunction2 = function (key, reducedVal) {
    reducedVal.avg = reducedVal.qty/reducedVal.count;
    return reducedVal; };
```

Beyond MapReduce

- Need to cover
  - More complex, multi-stages applications
  - Interactive ad-hoc queries
- Limitations of preexisting technology
  - Lack of abstractions for leveraging **distributed memory**
  - Reusing intermediate results across **multiple computations**
- Notion of RDD (Resilient Distributed Data)
  - High level operators
  - Distributed execution based on MapReduce
  - Notion of **RDD** (Resilient Distributed Datasets)

Conclusions about the state of data management systems

⇒ **Very active** research and development field [AAA+22]

- SQL-style APIs predominant to query and retrieve data
- Execution over a large cluster: shared-nothing, scale-out parallelism
- Columnar storage: widely used in most commercial data analytic platforms
- Memory-based data management systems
- Database systems offered as cloud services
- Hybrid transactional/analytical processing (HTAP) systems
- Modern database engine are based on sophisticated optimization techniques: memory-optimized data structures, modern compilation, code-generation, ...
- A new generation of data cleaning and data wrangling technology

- Emergence of data science: combines elements of data cleaning and transformation, statistical analysis, data visualization, and ML techniques
- New technical environment: notebooks, ...

What can data management do for machine learning?

- Minimize data movement
  - Avoid data duplication
  - Data inconsistency

  ⇒ **Program shipping vs. Data shipping**
- Efficient access and manipulation of data
  - Data layout, buffer management, indexing, data partitioning, parallel execution, . . .
  - **Automatic** query optimization
  - Metadata: schema information can help in modeling/data validation
- Predictions with data: **declarative machine learning**
- Security

What can data management do for machine learning?

- Minimize data movement
  - Avoid data duplication
  - Data inconsistency

  ⇒ **Program shipping vs. Data shipping**
- Efficient access and manipulation of data
  - Data layout, buffer management, indexing, data partitioning, parallel execution, ...
  - **Automatic** query optimization
  - Metadata: schema information can help in modeling/data validation
- Predictions with data: **declarative machine learning**
- Security

⇒ Machine learning in data management systems

ML in DBMS

Some challenges

- Abstractions: relational abstraction not enough
- Access Patterns: understanding how does an ML algorithm access data?
  Sequentially, randomly, repeated scans
- Automatic optimization: logical/physical optimization, cost model, ...
- New Data Types: Images, video, models, how do we store them and manage them?
- Parallel and distributed execution

ML in DBMS

Two selected topics

- ML system abstractions in data management system
- Declarative predictive queries

ML abstractions in data analysis tasks

- Descriptive analytics
  - Complex queries on a database system to extract aggregated information: statistics for a collection of records
  - Data abstraction: relation
  - Data processing operators : relational algebra
- Predictive analytics: study historical data in order to identify trends and produce predictions for future events
  - Machine learning algorithms for regression, classification and clustering
  - Data abstractions: matrix, vectors
  - Data processing operators: linear algebra
    An iterative refinement process to minimize/maximize a given objective function
  - Examples: linear/logistic regression, support vector machines (SVM), k-means, . . .

ML abstractions in data system

Matrix representation

- Representing large Vectors and Matrix as relations
- Goal: matrix storage and partitioning in a parallel system

```
A(row_number integer, vector numeric [])
A(row_number integer, column_number integer, value number)
```

ML abstractions in data system

Matrix representation

- Representing large Vectors and Matrix as relations
- Goal: matrix storage and partitioning in a parallel system

```
A(row_number integer, vector numeric [])
A(row_number integer, column_number integer, value number)
```

⇒ Horizontal partitioning by the DBMS (hashing, round-robin, . . .)

- Sparse vs. complete matrix

Basic matrix arithmetics

- Addition of two matrix A and B of identical dimensions

```
SELECT A.row_number, A.vector + B.vector
     FROM A, B WHERE A.row_number = B.row_number;
```

Basic matrix arithmetics

- Addition of two matrix A and B of identical dimensions

```
SELECT A.row_number, A.vector + B.vector
     FROM A, B WHERE A.row_number = B.row_number;
```

⇒ A query optimizer is likely to choose a hash join for this query (suitable for parallelization)

Basic matrix arithmetics

- Multiplication of a matrix and a vector Av

```
SELECT 1, array_accum(row_number, vector*v) FROM A;
```

  - * operator can be implemented as an UDF to express a dot product: $\vec{x}.\overrightarrow{y} = \sum_i x_i y_i$)
  - **array_accum(x,v)** is a UDAF which returns an array (setting position x to value v for each row of input)

Basic matrix arithmetics

- Multiplication of a matrix and a vector Av

```
SELECT 1, array_accum(row_number, vector*v) FROM A;
```

  - * operator can be implemented as an UDF to express a dot product: $\vec{x}.\overrightarrow{y} = \sum_i x_i y_i$)
  - **array_accum(x,v)** is a UDAF which returns an array (setting position x to value v for each row of input)

$\Rightarrow$ Parallelization thanks to the **Merge** function of the UDAF pattern

Basic matrix arithmetics

- Matrix transpose ($m \times n$))

```
SELECT S.col_number, array_accum(A.row_number,
A.vector[S.col_number]) FROM A, generate_series(1,3) AS
S(col_number) GROUP BY S.col_number;
```

Basic matrix arithmetics

- Matrix transpose ($m \times n$))

```
SELECT S.col_number, array_accum(A.row_number,
A.vector[S.col_number]) FROM A, generate_series(1,3) AS
S(col_number) GROUP BY S.col_number;
```

$\Rightarrow$ n copies of A are sent to the Group-By operator

Basic matrix arithmetics

- Matrix transpose ($m \times n$))

```
SELECT S.col_number, array_accum(A.row_number,
A.vector[S.col_number]) FROM A, generate_series(1,3) AS
S(col_number) GROUP BY S.col_number;
```

$\Rightarrow$ n copies of A are sent to the Group-By operator
$\Rightarrow$ Eager Group-by is useless

Basic matrix arithmetics

- Matrix transpose ($m \times n$))

```
SELECT S.col_number, array_accum(A.row_number,
A.vector[S.col_number]) FROM A, generate_series(1,3) AS
S(col_number) GROUP BY S.col_number;
```

$\Rightarrow$ n copies of A are sent to the Group-By operator
$\Rightarrow$ Eager Group-by is useless
$\Rightarrow$ Parallelization thanks to the **Merge** function of the UDAF pattern

Basic matrix arithmetics

- Matrix transpose ($m \times n$))

```
SELECT S.col_number, array_accum(A.row_number,
A.vector[S.col_number]) FROM A, generate_series(1,3) AS
S(col_number) GROUP BY S.col_number;
```

  - ⇒ *n* copies of *A* are sent to the Group-By operator
  - ⇒ Eager Group-by is useless
  - ⇒ Parallelization thanks to the **Merge** function of the UDAF pattern

- Matrix product
  - Which storage better fits ?

```
A(row_number integer, vector numeric [])
A(row_number integer, column_number integer, value
number)
```

Basic matrix arithmetics

- Matrix transpose ($m \times n$))

```
SELECT S.col_number, array_accum(A.row_number,
A.vector[S.col_number]) FROM A, generate_series(1,3) AS
S(col_number) GROUP BY S.col_number;
```

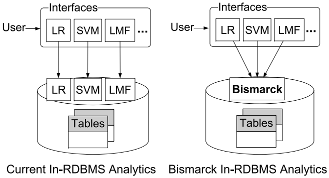
  ⇒ *n* copies of *A* are sent to the Group-By operator
  ⇒ Eager Group-by is useless
  ⇒ Parallelization thanks to the **Merge** function of the UDAF pattern

- Matrix product
  - Which storage better fits ?

```
A(row_number integer, vector numeric [])
A(row_number integer, column_number integer, value
number)
SELECT A.row_number, B.column_number, SUM(A.value *
B.value) FROM A, B WHERE A.column_number = B.row_number
     GROUP BY A.row_number, B.column_number
```

Beyond basic matrix arithmetics

Many ML techniques (mostly generalized linear models) can be reduced to mathematical programming and there is a single solver (Incremental Gradient Descent) that fits existing database system abstractions (User Defined Aggregates) [MR21b]



Analytics Task	Objective
Logistic Regression (LR)	$\sum_i \log(1 + exp(-y_i w^T x_i)) + \mu\|\vec{w}\|_1$
Classification (SVM)	$\sum_i (1 - y_i w^T x_i)_+ + \mu\|\vec{w}\|_1$
Recommendation (LMF)	$\sum_{(i,j)\in\Omega} (L_i^T R_j - M_{ij})^2 + \mu\|L, R\|_F^2$
Labeling (CRF) [48]	$\sum_k \left[\sum_j w_j F_j(y_k, x_k) - \log Z(x_k)\right]$
Kalman Filters	$\sum_{t=1}^T \|\|Cw_t - f(y_t)\|\|_2^2 + \|\|w_t - Aw_{t-1}\|\|_2^2$
Portfolio Optimization	$p^T w + w^T \Sigma w \quad \text{s.t.} \quad w \in \Delta$

- Unified implementation abstractions for in-data system ML: GLADE, MADlib, Spark MLlib, ...
- Active research on support for Deep Learning over DB-resident data

Prediction queries [PSB+22]

- Trained ML models are being deployed in a wide variety of scenarios
- High-value data in the enterprise is typically stored in relational databases, data warehouses or data lakes
- ML inference: prediction queries
  - ⇒ Complex analytics queries that employ trained pipelines to perform predictions over new data arriving in the database/data lake
  - ⇒ Prediction-specific logic implemented using data processing operators (e.g., filters or joins)

Prediction queries [PSB+22]

- Trained ML models are being deployed in a wide variety of scenarios
- High-value data in the enterprise is typically stored in relational databases, data warehouses or data lakes
- ML inference: prediction queries
  - ⇒ Complex analytics queries that employ trained pipelines to perform predictions over new data arriving in the database/data lake
  - ⇒ Prediction-specific logic implemented using data processing operators (e.g., filters or joins)

⇒ Optimizations spanning data and ML operators in prediction queries

Prediction queries

Example

- Model to predict whether a patient is in high risk of COVID-19 complications (covid_risk.onnx)
- Hospital data

  Patient_info
  Blood_test
  Pulmonary_test
- Prediction query

  Q: *"find asthma patients who are likely in the high-risk COVID-19 group"*

Prediction queries

Example

- Model to predict whether a patient is in high risk of COVID-19 complications (covid_risk.onnx)
- Hospital data

  Patient_info
  Blood_test
  Pulmonary_test
- Prediction query

  Q: *"find asthma patients who are likely in the high-risk COVID-19 group"*

$\Rightarrow$ Prediction-specific logic: Join, Invoke M, Filter

Prediction queries

The predict operator

$$
\begin{array}{l}
PREDICT \\
(\\
\qquad MODEL = model\ , \\
\qquad DATA = object\ AS\ <tables\_alias> \\
) \\
WITH\ (\ <result\_set\_definition>\ )
\end{array}
$$

Prediction queries

The predict operator

PREDICT
(
    MODEL = model ,
    DATA = object AS <tables_alias>
)
WITH (<result_set_definition>)

Select d.*, p.Score From PREDICT(MODEL = @model, DATA = dbo.mytable AS d) WITH (Score FLOAT) AS p;

Queries using the predict operator

Example

```
WITH data AS(
    SELECT *
    FROM patient_info AS pi
    JOIN pulmonary_test AS pt ON pi.id=pt.id
    JOIN blood_test AS bt ON pt.id=bt.id);
```

Queries using the predict operator

Example

```
WITH data AS(
      SELECT *
      FROM patient_info AS pi
      JOIN pulmonary_test AS pt ON pi.id=pt.id
      JOIN blood_test AS bt ON pt.id=bt.id);

SELECT d.id
      FROM PREDICT(MODEL = covid_risk.onnx,
                   DATA=data AS d)
      WITH(risk_of_covid float) AS p
      WHERE d.asthma = 1 AND p.risk_of_covid = "high";
```

Holistic optimization of prediction queries [PSB+22]

- Unified Internal Representation (IR)
  - Relational algebra
  - Linear algebra
  - Other ML operators and data featurizers (e.g., decision trees, categorical encoding, text featurization, ...)
- Optimization
  - Logical optimizations
  - Logical-to-physical optimizations
- Execution of optimized plans
  - Apache Spark or SQL Server (+ ONNX Runtime)

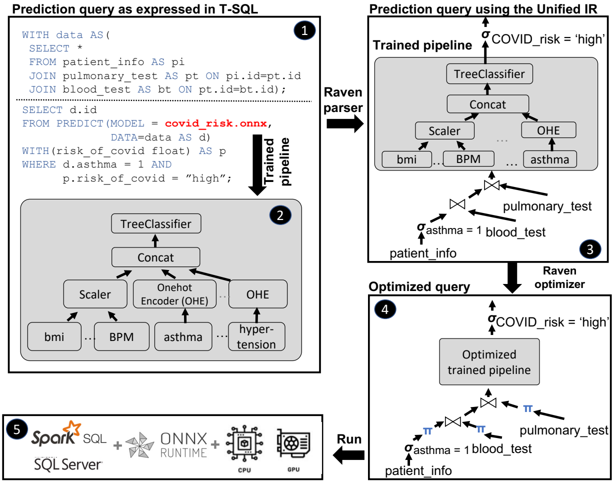
Holistic optimization of prediction queries [PSB+22]

- Unified Internal Representation (IR)
  - Relational algebra
  - Linear algebra
  - Other ML operators and data featurizers (e.g., decision trees, categorical encoding, text featurization, ...)

    Arbitrary algorithms !!
- Optimization
  - Logical optimizations
  - Logical-to-physical optimizations
- Execution of optimized plans
  - Apache Spark or SQL Server (+ ONNX Runtime)

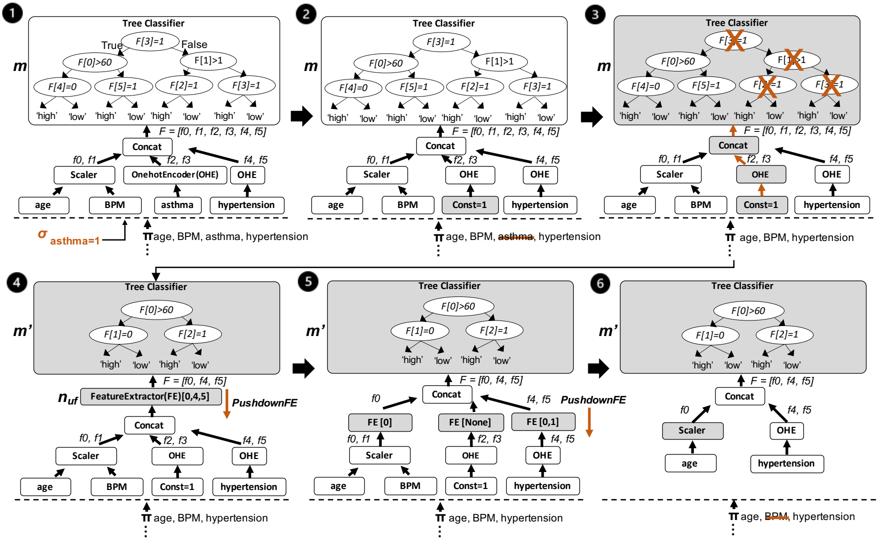
End-to-End optimization of prediction queries

Example



Raven logical optimizations

- Cross-optimizations
  - ⇒ Predicate-based model pruning
  - ⇒ Model-projection pushdown
- Data-induced optimizations
  - Using data statistics to optimize ML models of prediction queries: pruning subtrees based on data distribution
  - Sharding/data partioning: Compiling optimized model for each partition


1
Tree Classifier
m
F[3]=1
True
False
F[0]>60
F[1]>1
F[4]=0
F[5]=1
F[2]=1
F[3]=1
'high' 'low' 'high' 'low' 'high' 'low' 'high' 'low'
F = [f0, f1, f2, f3, f4, f5]
Concat
f0, f1
f2, f3
f4, f5
Scaler
OnehotEncoder(OHE)
OHE
age
BPM
asthma
hypertension
σ asthma=1
π age, BPM, asthma, hypertension
2
Const=1
π age, BPM, ~~asthma~~, hypertension
3
π age, BPM, hypertension
4
m'
F[0]>60
F[1]=0
F[2]=1
F = [f0, f4, f5]
n_uf
FeatureExtractor(FE)[0,4,5]
PushdownFE
5
FE [0]
FE [None]
FE [0,1]
6
π age, ~~BPM~~, hypertension

Logical-to-physical optimizations

- Transformations for Runtime Selection
  - MLtoSQL: turns ML operators to SQL statements
    - ⇒ . Reduce/eliminate the invocation of ML runtime: avoiding initialization costs and data conversions/copies between the relational and ML engines
    - ⇒ Enables more extensive relational optimizations in the DBMS
  - MLtoDNN: transforms traditional ML operators to equivalent (deep) neural networks (DNN)
- Data-driven optimizations

MLtoSQL

- Linear models and scaling operators → multiplication/addition/subtraction operators
- Tree-based models and encoding operators → case statements

```
CASE WHEN F[0] > 60 THEN (
    CASE WHEN F[1] = 0 THEN 1 ELSE 0 END) ELSE (
    CASE WHEN F[2] = 1 THEN 1 ELSE 0 END) END
```

Conclusion

- ML is moving to a pervasive technology
- ML models are used by expert developers working within large organizations
- Next wave of ML systems: allow a larger amount of people, potentially without coding skills, to perform the same tasks

$\Rightarrow$ Needs for declarative interfaces

- Generations data management (but also compiler, operating systems, software engineering) work may inspire new foundational questions
- Challenging issues from the data management perspective

References I

Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip A. Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, Anhai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca Popa, Raghu Ramakrishnan, Christopher Re, Michael Stonebraker, and Dan Suciu, *The seattle report on database research*, Commun. ACM **65** (2022), no. 8, 72?79.

References II

- Sara Cohen, *User-defined aggregate functions: Bridging theory and practice*, SIGMOD '06 (New York, NY, USA), ACM, 2006, pp. 49–60.
- Seth Gilbert and Nancy Lynch, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*, SIGACT News **33** (2002), no. 2, 51–59.
- César A. Galindo-Legaria and Milind Joshi, *Orthogonal optimization of subqueries and aggregation*, ACM SIGMOD Conference, 2001.
- Piero Molino and Christopher Ré, *Declarative machine learning systems*, Commun. ACM **65** (2021), no. 1, 42?49.

References III

- ______, *Declarative machine learning systems*, Commun. ACM **65** (2021), no. 1, 42–49.

- Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos, *End-to-end optimization of machine learning prediction queries*, Proceedings of the 2022 International Conference on Management of Data (New York, NY, USA), SIGMOD '22, Association for Computing Machinery, 2022, p. 587–601.

- Karthik Ramachandra, Kwanghyun Park, K. Venkatesh Emani, Alan Halverson, César Galindo-Legaria, and Conor Cunningham, *Froid: Optimization of imperative programs in a relational database*, Proc. VLDB Endow. **11** (2017), no. 4, 432–444.

References IV

- Varun Simhadri, Karthik Ramachandra, Arun Chaitanya, Ravindra Guravannavar, and S. Sudarshan, *Decorrelation of user defined function invocations in queries*, IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014 (Isabel F. Cruz, Elena Ferrari, Yufei Tao, Elisa Bertino, and Goce Trajcevski, eds.), IEEE Computer Society, 2014, pp. 532–543.
- Weipeng P. Yan and Per-Åke Larson, *Eager aggregation and lazy aggregation*, Very Large Data Bases Conference, 1995.