Algorithms and Optimizations for **Big Data** Analytics: Cubes

Carlos Ordonez
University of Houston
USA

Goals of talk

- State of the art in large-scale analytics, including big data
- Contrast SQL/UDFs and MapReduce
- Collaboration on new projects

Global Outline

1. Data mining models and algorithms
   - 1.1 Analytics: statistics, cubes, statistical models
   - 1.2 Data set
   - 1.3 Models and Algorithms
   - 1.4 Big data
2. Processing alternatives
   - 2.1 Inside DBMS: SQL and UDFs
   - 2.2 Outside DBMS: MapReduce, C++
   - 2.3 Optimizations
3. Research Directions

Analytics

- Simple:
  - Ad-hoc Queries
  - **Cubes**: OLAP, MOLAP, includes descriptive statistics: histograms, means, plots, statistical tests
- Complex:
  - Models
  - Patterns

Data set

- Data set F : n records, d dimensions , e measures
- Dimensions: discrete, measures: numeric
- Focus of the talk, d dimensions $D = \{D_1, \ldots, D_d\}$
- I/O bottleneck: $d \ll n$ $A = \{A_1, A_2, \ldots, A_e\}$
- Cube: the lattice of d dimensions
- High d makes problem computionally more difficult

Cube computations

- Large *n*: F cannot fit in RAM, minimize I/O
- Multidimensional
  - *d*: tens (hundreds?) of dimensions
- Computed with data structures

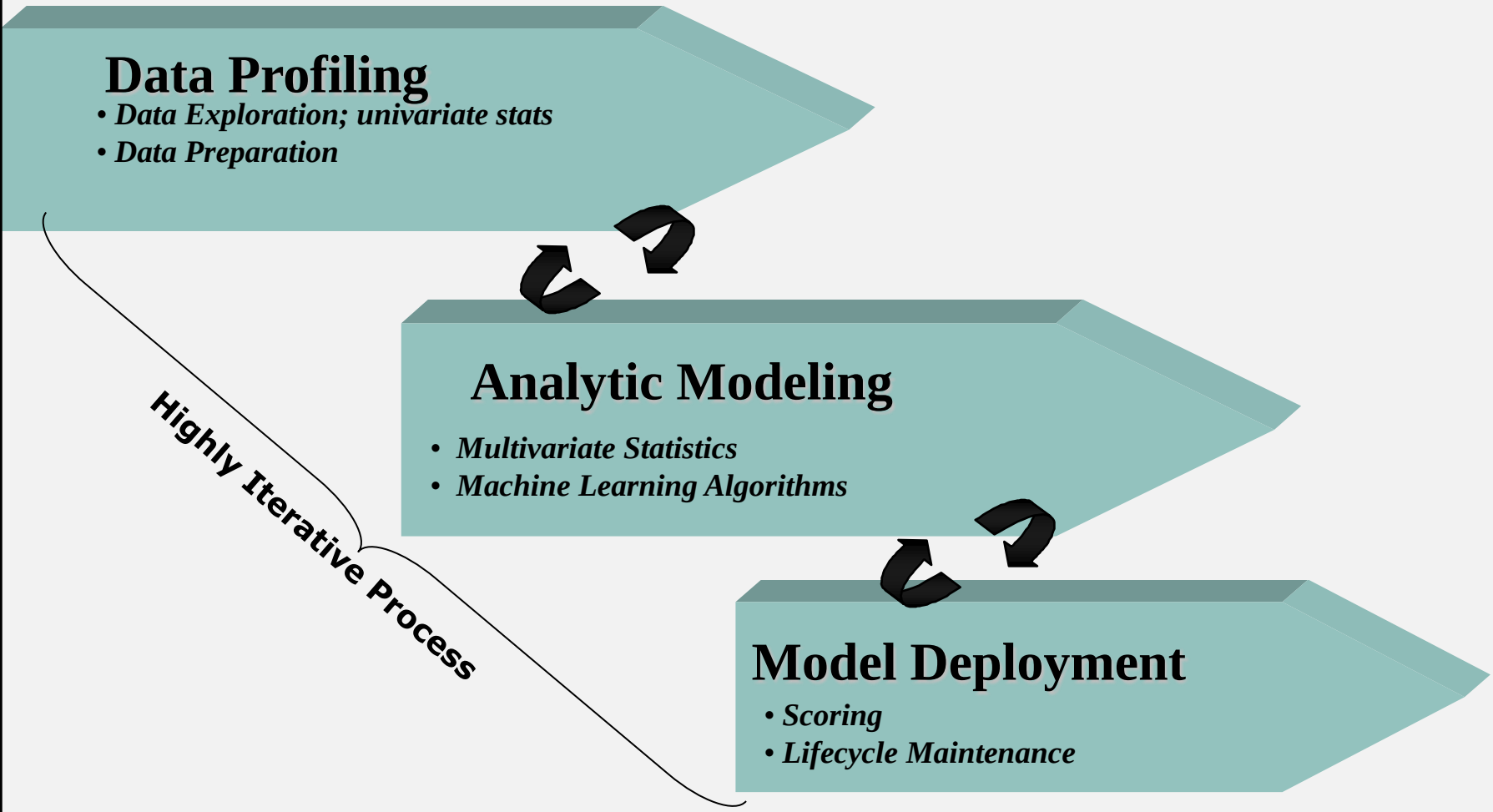
Algorithms

- Behavior with respect to data set X:
  - Level-wise: k passes
- Time complexity: $O(n2^d)$
- Research issues:
  - Parallel processing
  - different time complexity in SQL/MapReduce
  - Incremental and online learning

Analytics

1. Prepare and clean data
2. Explore data set: cubes and descriptive statistics
3. Model computation
4. Scoring and model deployment

Analytics Process: big data?



Some overlooked aspects

- Preparing and cleaning data takes a lot of time.: ETL
- Lots of SQL written to prepare data sets for statistical analysis
- Data quality was hot; worth revisiting w/big data
- Strong emphasis on large n in data mining
- Cube computation is the most researched topic; cube result analysis/interpretation 2nd priority
- Big data different?

SQL to ER

- Goal: creating a data set *X* with *d* dimensions *D(K,A)*, *K* commonly a single id
- Lots of SQL queries, many temporary tables
- Decoupled from ER model, not reused
- Many transformations: cubes, variable creation

SQL to ER

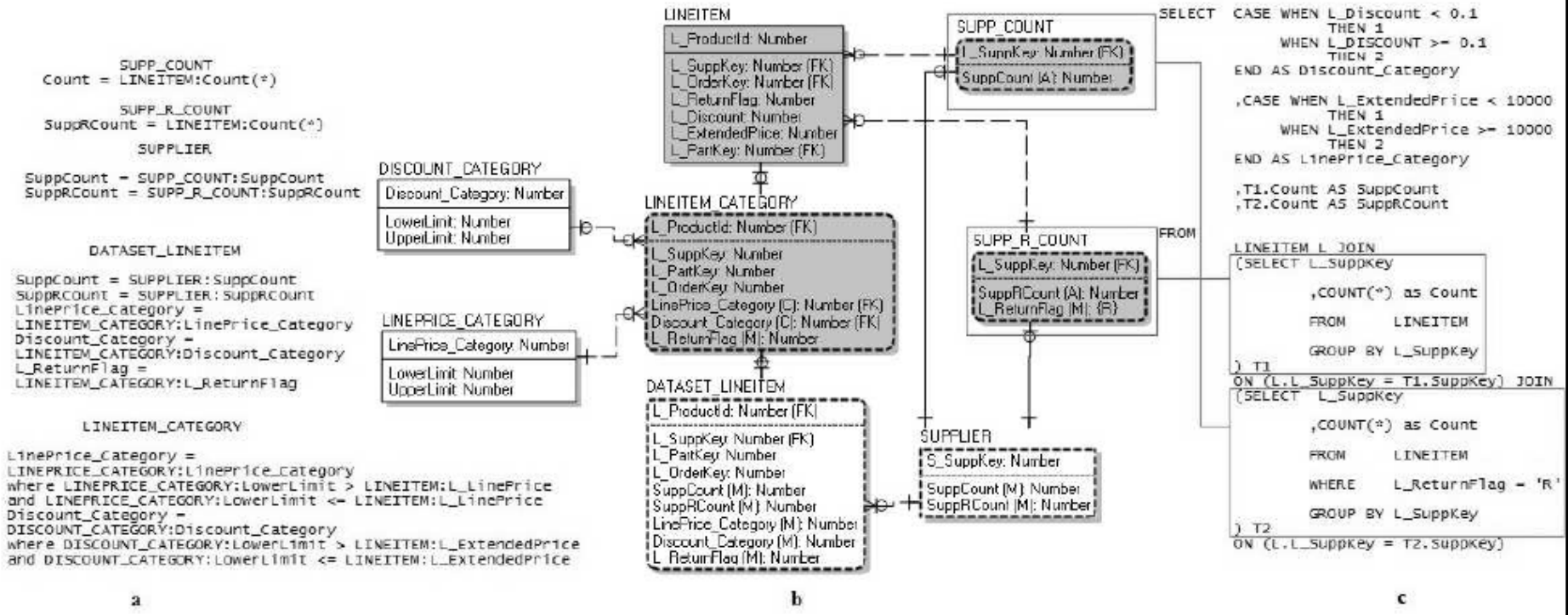
```
SELECT  CASE A1 WHEN 1 THEN 'A'
                ELSE 'B'
        END AS DerAtt1
        ,CASE WHEN (A1/2 > 2) THEN 'X'
                ELSE 'Y'
        END AS DerAtt2

FROM    T1
```

a

```
SELECT  A1
        ,SUM(A2) AS DerAtt3
FROM    T2
GROUP BY A1
```

b

SQL transformations in ER Example with TPC-H



Horizontal aggregations

- Create cross-tabular tables from cube
- PIVOT requires knowing values
- Aggregations in horizontal layout

Horizontal aggregations on cube

F

K	D_1	D_2	D_3	A_1
1	2	a	x	2
2	2	b	y	2
3	4	b	z	4
4	2	a	x	0
5	4	c	y	0
6	2	a	z	1
7	1	c	x	**null**
8	2	b	y	1
9	2	a	x	2
10	3	a	y	8

F_V

D_1	D_2	count(*)	A
1	c	1	**null**
2	a	4	5
2	b	2	3
3	a	1	8
4	b	1	4
4	c	1	0

F_H

D_1	sum(A BY D_2=a)	sum(A BY D_2=b)	sum(A BY D_2=c)
1	null	null	**null**
2	5	3	null
3	8	null	null
4	null	4	0

```
/* F_V */
SELECT D1, D2,count(*),sum(A)
FROM F
GROUP BY D1, D2
ORDER BY D1, D2;

/* F_H */
SELECT D1,sum(A BY D2)
FROM F
GROUP BY D1
ORDER BY D1;
```

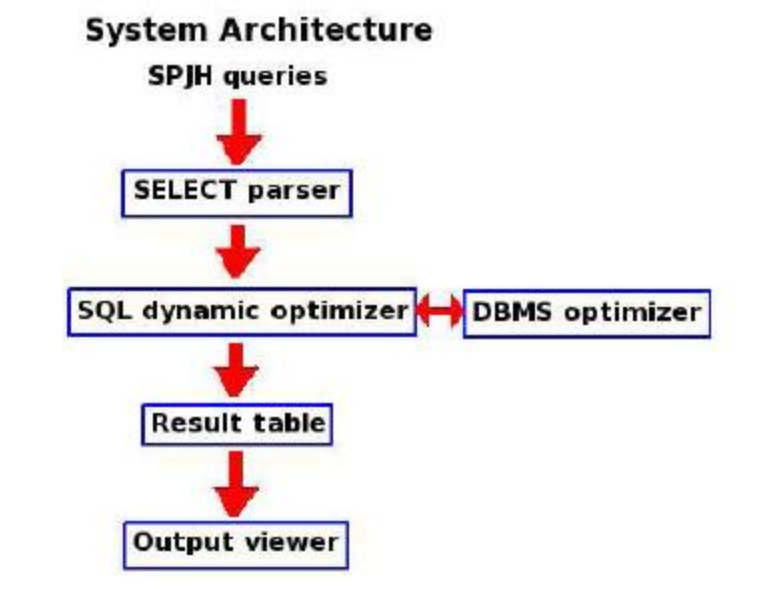
Prepare Data Set
Horizontal aggregations

```
SELECT
   storeId,
   sum(salesAmt BY dayofweekName),
   sum(salesAmt)
FROM transactionLine
     ,DimDayOfWeek,DimMonth
WHERE transactionLine.dayOfWeekNo
      =DimDayOfWeek.dayOfWeekNo
GROUP BY storeId;
```

store Id	salesAmt							total sales
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
1	500	200	120	140	90	230	160	1440
2	200	100	400	100	900	100	200	2000
3	100	100	100	200	200	200	200	1100
4	200	300	200	300	200	300	200	2700

Figure 1: A tabular data set obtained from table *transactionLine*

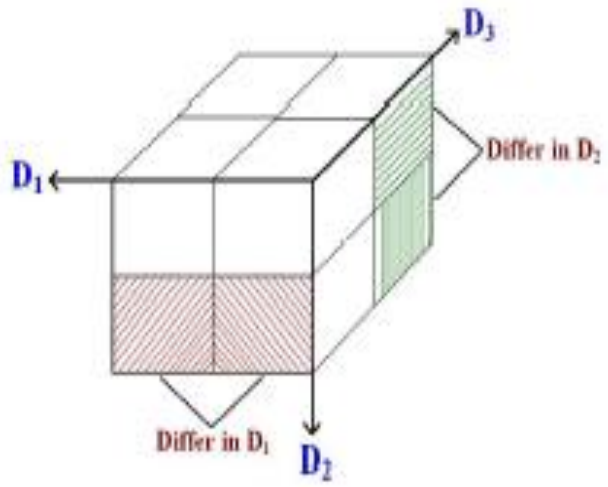
Metaoptimizer

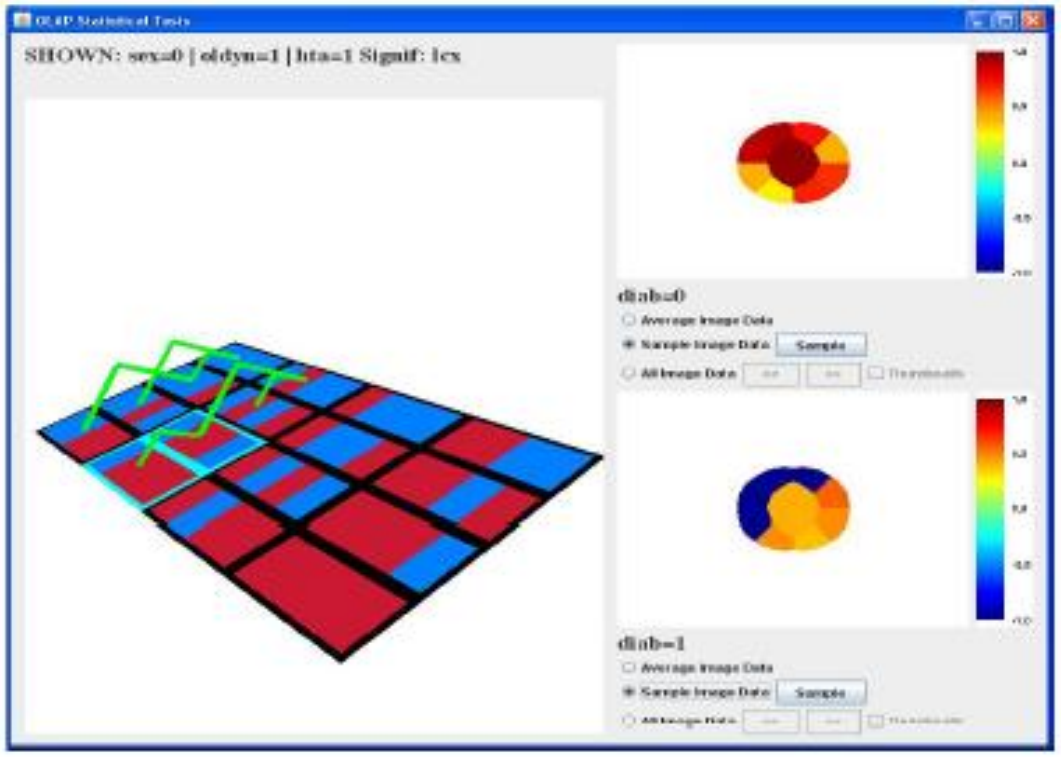


Cube visualization

- Lattice exploration
- Projection into 2D
- Comparing cuboids

Cube interpretation & visualization statistical tests on cubes





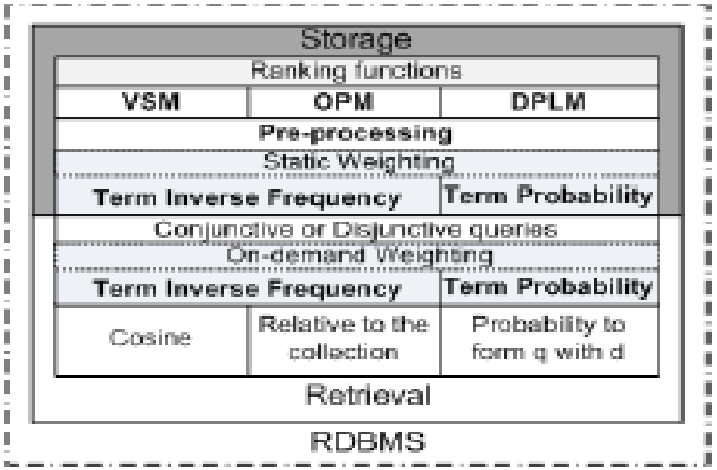
Big data

- Finer granularity than transactions
- In general big data cannot be directly analyzed: pre-processing needed
- Diverse data sources, non-relational, beyond alphanumeric
- Web logs, user interactions, social networks, streams

Issues about big data

- NoSQL, no DDL
- Transaction processing
- Web-scale data is not universal
- Many (most?) practical problems are smaller
- Database integration and cleaning much harder
- Parallel processing is becoming a standard
- SQL remains query alternative

Big data
IR: Keyword search, ranking



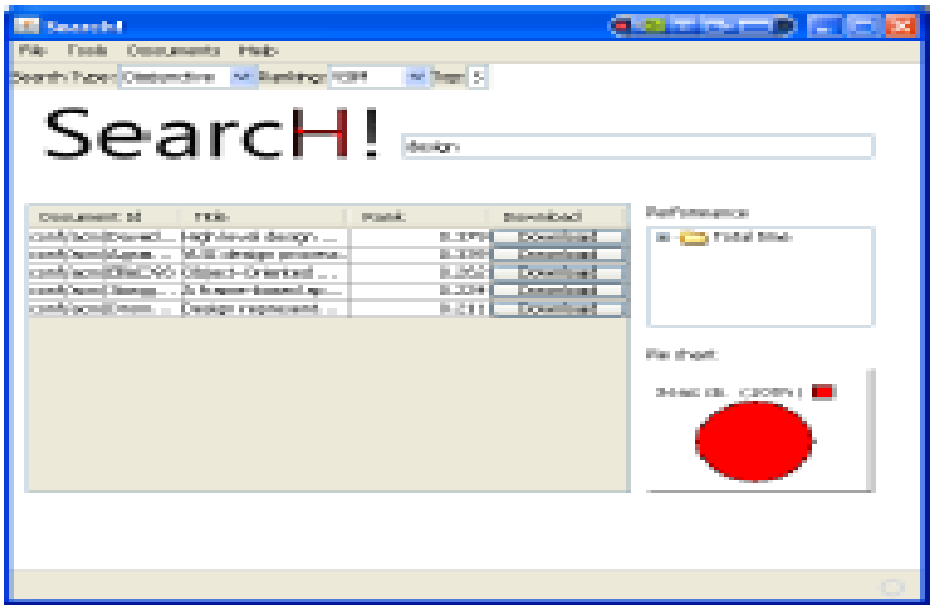


Fig. 4. Document Retrieval

2. Processing alternatives

2.1 Inside DBMS (SQL)

2.2 Outside DBMS (MapReduce,brief review of processing in C, external packages)

2.3 Optimizations

Why mining inside the DBMS?

- Huge data volumes: potentially better results with larger amounts of data; less process. time
- Minimizes data redundancy; Eliminate proprietary data structures; simplifies data management; security
- Caveats: SQL, limited statistical functionality, complex DBMS architecture

2.1 Inside DBMS

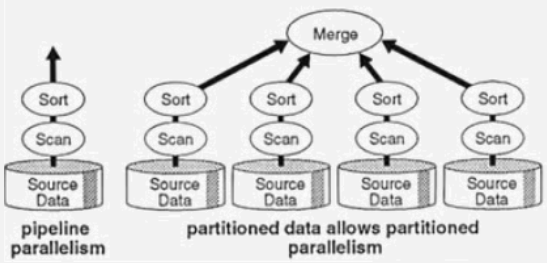
- Assumption:
  - data records are in the DBMS; exporting slow
  - row-based storage (not column-based)
- Programming alternatives:
  - SQL and UDFs: SQL code generation (JDBC), precompiled UDFs. Extra: SP, embedded SQL, cursors
  - Internal C Code (direct access to file system and mem)
- DBMS advantages:
  - important: storage, queries, security
  - maybe: recovery, concurrency control, integrity, transactions

Inside DBMS

Physical Operators

[DG1992,CACM] [SMAHHH2007,VLDB] [WH2009,SIGMOD]

- Serial DBMS (one CPU, RAID):
  - table Scan
  - join: hash join, sort merge join, nested loop
  - external merge sort
- Parallel DBMS (shared-nothing):
  - even row distribution, hashing
  - parallel table scan
  - parallel joins: large/large (sort-merge, hash); large/short (replicate short)
  - distributed sort



Inside DBMS

User-Defined Function (UDF)

- Classification:
  - Scalar UDF
  - **Aggregate UDF**
  - Table UDF
- Programming:
  - Called in a SELECT statement
  - C code or similar language
  - API provided by DBMS, in C/C++
  - Data type mapping

Inside DBMS

UDF pros and cons

- Advantages:
  - arrays and flow control
  - Flexibility in code writing and no side effects
  - No need to modify DBMS internal code
  - In general, simple data types
- Limitations:
  - OS and DBMS architecture dependent, not portable
  - No I/O capability, no side effects
  - Null handling and fixed memory allocation
  - Memory leaks with arrays (matrices): fenced/protected mode

Inside DBMS

Aggregate UDF (skipped scalar UDF)

[JM1998,SIGMOD]

- Table scan
- Memory allocation in the heap
- GROUP BY extend their power
- Also require handling nulls
- Advantage: parallel & multithreaded processing
- Drawback: returns a single value, not a table
- DBMSs: SQL Server, PostgreSQL,Teradata, Oracle, DB2, among others
- Useful for model computations

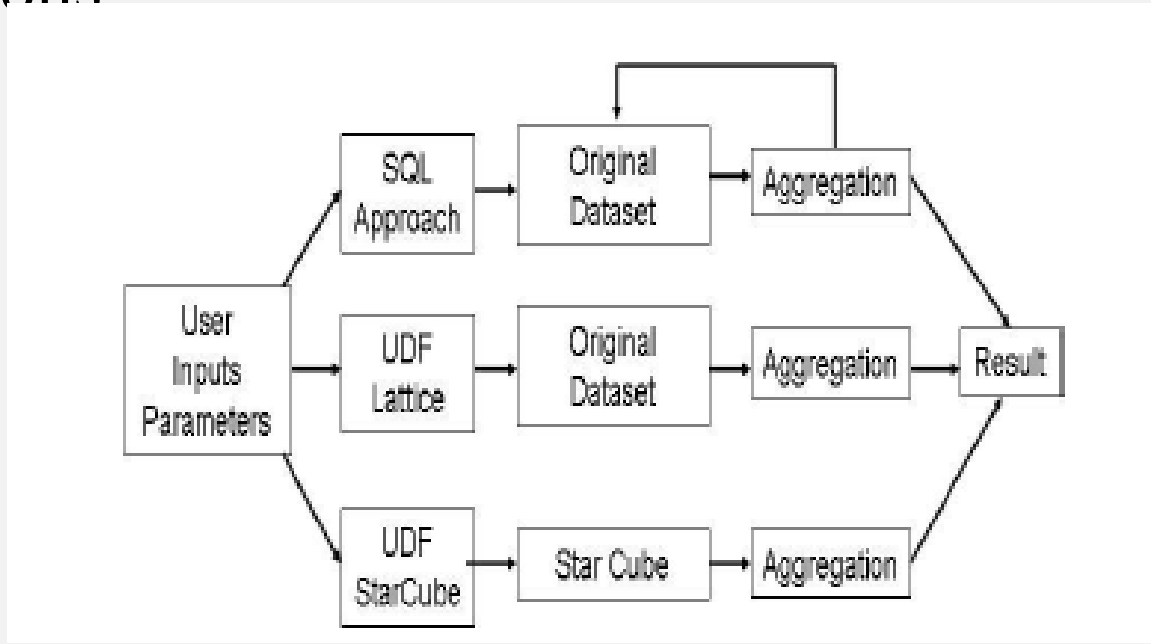
Inside DBMS

Table UDF

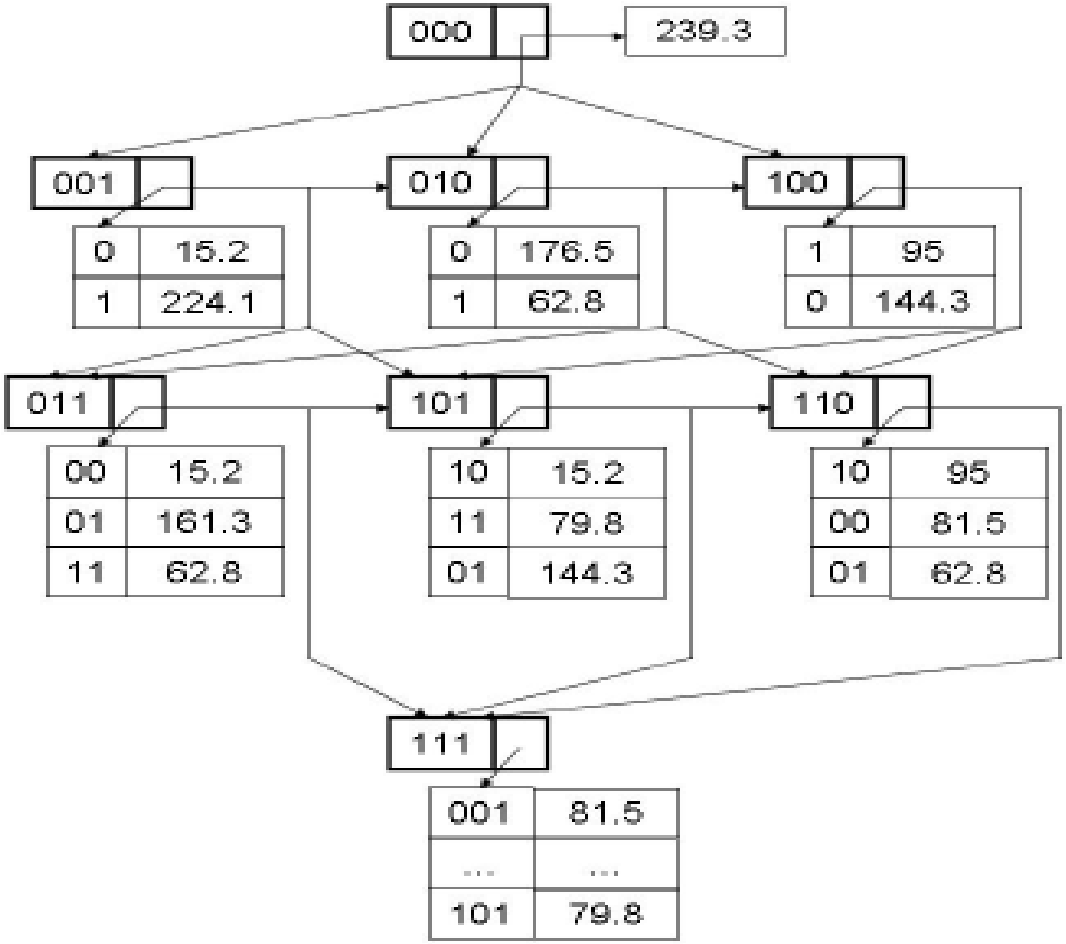
- Main difference with aggregate UDF: returns a table (instead of single value)
- Also, it can take several input values
- Called in the FROM clause in a SELECT
- Stream: no parallel processing, external file
- Computation power same as aggregate UDF
- Suitable for complex math operations and algorithms
- Since result is a table it can be joined

Cube computation with UDF (table function)

- Data structure in RAM; maybe one pass
- It requires maximal cuboid or choosing k dimensions



Cube in UDF
Lattice manipulated with hash table



Inside DBMS

Internal C code (if code available); not popular

- Advantages:
  - access to file system (table record blocks),
  - physical operators (scan, join, sort, search)
  - main memory, data structures, libraries
  - hardware: multithreading, multicore CPU, RAM, caching LI/L2
  - LAPACK
- Disadvantages:
  - requires careful integration with rest of system
  - not available to end users and practitioners
  - may require exposing functionality with DM language or SQL

Outside DBMS

MapReduce

[DG2008,CACM]

- Parallel processing; simple; shared-nothing
- Commodity diverse hardware (big cluster)
- Functions are programmed in a high-level programming language (e.g. Java, Python); flexible.
- <*key*,*value*> pairs processed in two phases:
  - map(): computation is distributed and evaluated in parallel; independent mappers
  - reduce(): partial results are combined/summarized
- Can be categorized as inside/outside DBMS, depending on level of integration with DBMS

Outside DBMS: alternatives

Packages, libraries, Java/C++

[ZHY2009,CIDR] [ZZY2010,ICDE]

- MOLAP tools:
  - Push aggregations with SQL
  - Memory-based lattice traversal
  - Interaction with spreadsheets
- Programming languages:
  - Arrays
  - flexibility of control statements
- Packages: Microstrategy, Business Objects

Optimization: Data Set Storage layout: Horizontal/Vertical

Horizontal	Vertical
Limitation with high d (max columns).	No problems with high *d*.
Default layout for most algorithms.	Requires clustered index.
SQL arithmetic expressions and UDFs.	SQL aggregations, joins, UDFs.
Easy to interpret.	Difficult to interpret.
Suitable for dense matrices.	Suitable for sparse matrices.
Complete record processing	UDF: detect point boundaries
n rows, *d* columns	*dn* rows, few (3 or 4) columns
Fast *n* I/Os	Slow *dn* I/Os (*n* I/Os clustered)

Optimizations
Algorithmic & Systems

- Algorithmic
  - 90% research, many efficient algorithms
  - accelerate/reduce cube computations
  - database systems focus: reduce I/O passes
  - approximate solutions
  - parallel
- Systems (SQL, MapReduce)
  - Platform: parallel DBMS server vs cluster of computers vs multicore CPUs
  - Programming: SQL/C++ versus Java

Algorithmic

- Programming: man times binary file required for random access
- data structures working in main memory and disk
- Programming not in SQL: C/C++ are preferred languages

Algorithmic Optmization: summary matrices

$$L = \begin{bmatrix} \sum X_1 \\ \sum X_2 \\ \vdots \\ \sum X_d \end{bmatrix}$$

$$Q = \begin{bmatrix} \sum X_1^2 & \sum X_1 X_2 & \dots & \sum X_1 X_d \\ \sum X_2 X_1 & \sum X_2^2 & \dots & \sum X_2 X_d \\ \vdots & & & \vdots \\ \sum X_d X_1 & \sum X_d X_2 & \dots & \sum X_d^2 \end{bmatrix}$$

$$\rho_{ab} = \frac{nQ_{ab} - L_a L_b}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}}$$

$$V_{ab} = \frac{1}{n} Q_{ab} - \frac{1}{n^2} L_a L_b.$$

$$C_j = \frac{1}{N_j} L_j,$$

$$R_j = \frac{1}{N_j} Q_j - \frac{1}{N_j^2} L_j L_j^T.$$

$$Q' = \begin{bmatrix} Q_{11} & Q_{12} & \dots & (XY^T)_1 \\ Q_{21} & Q_{22} & \dots & (XY^T)_2 \\ \vdots & \vdots & & \vdots \\ Q_{(d+1)1} & Q_{(d+1)2} & \ddots & (XY^T)_{d+1} \\ (XY^T)_1 & (XY^T)_2 & \dots & YY^T \end{bmatrix}$$

$$Q' : \ \beta = Q^{-1}(XY^T)$$

Link Statistical Models and Patterns: Cubes?

$$W = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix} \quad C = \begin{bmatrix} 0.00 & 0.66 \\ 1.00 & 0.66 \\ 0.50 & 0.33 \\ 1.00 & 0.00 \end{bmatrix} \quad R = \begin{bmatrix} 0.00 & 0.22 \\ 0.00 & 0.22 \\ 0.25 & 0.22 \\ 0.00 & 0.00 \end{bmatrix}$$

Figure 3: Clustering model; $d = 4, k = 2$.

		C_j dimension range for item i_l				
j	W_j	0.8-1	0.6-0.8	0.4-0.6	0.2-0.4	0-0.2
1	0.4	2 4		3		1
2	0.6		1 2		3	4

Figure 4: Clustering model summary.

$$\rho = \begin{bmatrix} 1 & & & \\ -0.61 & 1 & & \\ 0.16 & -0.61 & 1 & \\ -0.67 & -0.40 & 0.17 & 1 \end{bmatrix}$$

Figure 6: Correlation matrix.

DBMS SQL Optimizations

- SQL query optimization
  - mathematical equations as queries
  - Turing-complete: SQL code generation and programming language
- UDFs as optimization
  - substitute difficult/slow math computations
  - push processing into RAM memory

DBMS Query Optimizations

- Split queries; query optimizer falls short
- Join:
  - denormalized storage: model, intermediate tables
  - favor hash joins over sort-merge for data set
  - secondary indexing for join: sort-merge join
- Aggregation (create statistical variables):
  - push group-by before join: watch out nulls and high cardinality columns
  - Outer joins
- synchronized table scans: share I/O
- Sampling O(s) access, truly random; error

Systems Optimization

DBMS UDF

[HLS2005,TODS] [O2007,TKDE]

- UDFs can substitute SQL code
  - UDFs can express complex matrix computations
  - Scalar UDFs: vector operations
- Aggregate UDFs: compute data set summaries in parallel, especially sufficient statistics n,L,Q
- Table UDFs: streaming model; external temporary file; get close to array functionality

MapReduce Optimizations

[ABASR2009,VLDB] [CDDHW2009,VLDB] [SADMPPR2010,CACM]

- Data set
  - keys as input, partition data set
  - text versus sequential file
  - loading into file system may be required
- Parallel processing
  - high cardinality keys: *i*
  - handle skewed distributions
  - reduce row redistribution in Map( )
- Main memory processing

MapReduce

Common Optimizations

[DG2008,CACM] [FPC2009,PVLDB] [PHBB2009,PVLDB]

- Modify Block Size; Disable Block Replication
- Delay reduce(), chain Map()
- Tune M and R (memory allocation and number)
- Several M use the same R
- Avoid full table scans by using subfiles (requires naming convention)
- combine() in map() to shrink intermediate files
- SequenceFiles as input with custom data types.

MapReduce
Issues

- Loading, converting to binary may be necessary
- Not every analytic task is efficiently computed with MapReduce
- Input key generally OK if high cardinality
- Skewed map key distribution
- Key redistribution (lot of message passing)

Systems optimizations

SQL vs MR (optimized versions, run same hardware)

Task	SQL	UDF	MR
Speed: compute model	1	2	3
Speed: score data set	1	3	2
Programming flexibility	3	2	1
Process non-tabular data	3	2	1
Loading speed	1	1	2
Ability to add optimizations	2	1	3
Manipulating data key distribution	1	2	3
Immediate processing (push=SQL,pull=MR)	2	1	3

SQL versus MapReduce

Task	SQL	MR
Sequential open-source	y	y
Parallel open source	n	y
Fault tolerant on long jobs	n	Y
Libraries	limited	Many
Arrays and matrices	limited	good
Massive parallelism, large N	n	y

Research Issues

SQL and MapReduce

- Fast data mining algorithms solved? Yes, but not considering data sets are stored in a DBMS
- Big data is a rebirth of data mining
- SQL and MR have many similarities: shared-nothing
- New analytic languages
- Fast load/unload interfaces between both systems; tighter integration
- General tradeoffs in speed and programming: horizontal vs vertical layout
- Incremental algorithms: one pass (streams) versus parallel processing; reduce passes/iterations

Acknowledgments

- My PhD students
- NSF
- UH Department of Computer Science
  - Systems group
  - Analytics group
- Collaborators:
  - UH: PT Tosic, AI;
  - Mexico: J Garcia UNAM, Mexico
  - MDACC: VB Baladandayuthapani, MD Anderson,Statistics
  - J Hellerstein, UC Berkeley; D Srivastava, ATT Labs; C Jermaine, Rice U; O Frieder Georgetown U