*Integrating Big Data and Relational Data with a Functional SQL-Like Query Language**

Patrick Valduriez

(with Carlyna Bondiombouy, Boyan Kolev, Oleksandra Levchenko)

Inria, Montpellier, France

**Int. Conf. on Databases and Expert Systems Applications (DEXA), 2015*

BDEC: from simulation to analytics*

- Simulation: traditional HPC market
  - e.g. weather forecasting, climate modeling
- HPC applications need to deal with more and more data
  - More input data, e.g. from more powerful scientific instruments or sensor networks
  - More output data, e.g. with smarter mathematical models and algorithms
- Analytics: a newer, complementary market for HPC
  - Analytics methods applied to established HPC domains in industry, government, academia
  - High-end commercial analytics pushing up into HPC
  - Shorter journey from science to business, e.g. from cancer genomics to personalized medicine

*Big Data and Exascale Computing (BDEC) meeting, Barcelona, January 2015

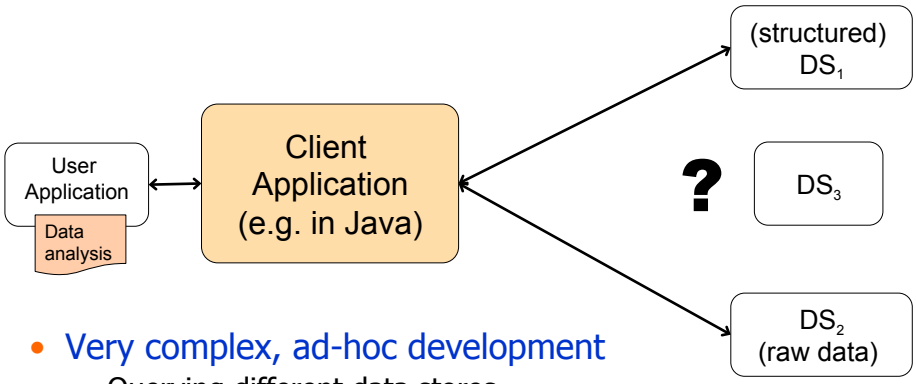
Big Data Analytics (BDA)

- Objective: find useful information and discover knowledge in data
  - Typical uses: forecasting, decision making, research, science, …
  - Techniques: data analysis, data mining, machine learning, …
- Why is this hard?
  - External data from various sources
    - Hard to verify and assess, hard to integrate
  - Different structures
    - Unstructured text, semi-structured document, key/value, table, array, graph, stream, time series, etc.
    - Hard to integrate
  - Low information density (unlike in corporate data)
    - Like searching for needles in a haystack
  - Simple machine learning models don't work
    - Reality and context matter
      - See "When big data goes bad" stories

Some BDA Killer Apps

- Real-time processing and analysis of raw data from high-throughput scientific instruments
  - E.g. to detect changing external conditions
- Uncertainty quantification in data, models, and experiments
  - E.g. to measure the reliability of simulations involving complex numerical models
- Fraud detection across massive databases
  - Applicable in many domains (e-commerce, banking, telephony, etc.)
- National security
  - Signal intelligence (SIGINT), anomaly detection, cyber analytics
  - Anti-terrorism (including evacuation planning), anti-crime
- Health care/medical science
  - Drug design, personalized medicine
  - Epidemiology
  - Systems biology
- Social network analysis
  - Modeling, simulation, visualization of large-scale networks

General Problem We Address Here



- Very complex, ad-hoc development
  - Querying different data stores
  - Managing intermediate results
  - Delivering (e.g. sorting) the final results
- Hard to extend
  - What if we want to access a new DS?

Our Approach: think declarative!

- Relational-like query language and generic query processing
  - Eases application development
  - Yields optimization and parallelization
- Some of previous related work in HOSCAR
  - E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, M. Mattoso. An Algebraic Approach for Data-centric Scientific Workflows. PVLDB, 4(12):1328-1339, 2011.
  - E. Ogasawara, J. Dias, V. Silva, F. Chirigati, D. de Oliveira, F. Porto, P. Valduriez, M. Mattoso. Chiron: A Parallel Engine for Algebraic Scientific Workflows. Concurrency and Computation: Practice and Experience, 25(16):2327-2341, 2013.
  - J. Dias, E. Ogasawara, D. de Oliveira, F. Porto, P. Valduriez, M. Mattoso. Algebraic Dataflows for Big Data Analysis. IEEE Int. Conf. on Big Data, 150-155, 2013.
  - J. Dias, G. Guerra, F. Rochinha, A. Coutinho, P. Valduriez, M. Mattoso. Data-Centric Iteration in Dynamic Workflows. Future Generation Computer Systems, Vol. 4, 114-126, 2015.

Outline

- Introduction
- Related work
- Contributions
- SQL-like query language
- Query engine
- Query rewriting
- Validation
- Conclusion

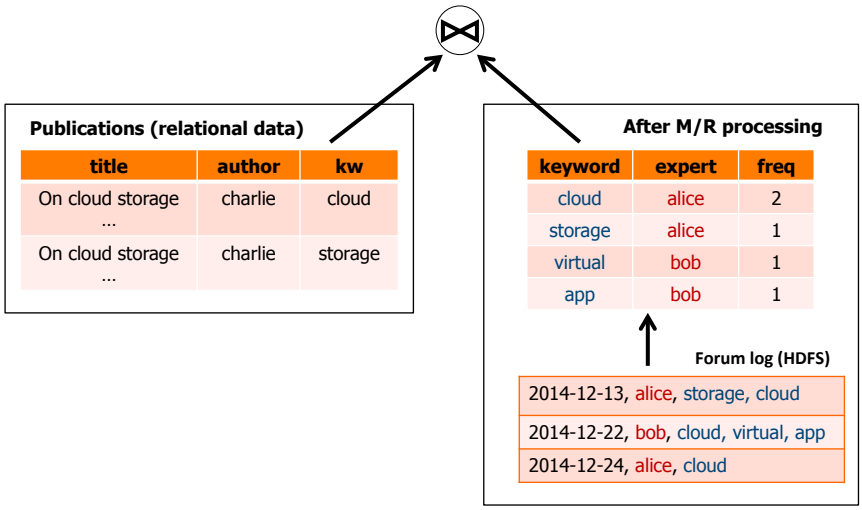
Introduction

- Context: *multistores*
  - Also called *polystores* *
    - Provide access to multiple cloud data stores such as NoSQL, HDFS and RDBMS
  - CoherentPaaS multistore's CloudMdsQL language
- Objectives
  - Integration of structured (relational) and unstructured (HDFS) data
  - Extend the functionality of CloudMdsQL to query data stored in HDFS using a data processing framework (DPF) like MapReduce or Spark

*M. Stonebraker. The Case for Polystores. ACM SIGMOD blog. July 2015.

Motivating Example

An editorial office needs to find appropriate reporters for a list of publications based on given keywords



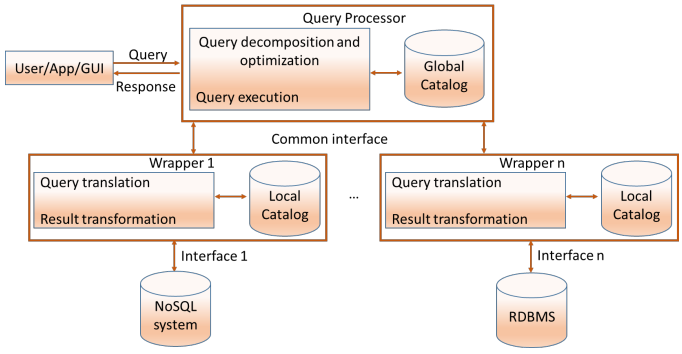
Contributions

- Our approach: CloudMdsQL
  - Functional SQL-like query language and query engine
    - Simple notation
      - MFR subquery = sequence of Map/Filter/Reduce operators
  - Integration of structured and unstructured data stores
    - Unstructured data transformed in data processing framework
  - Query optimization techniques
    - Selection pushdown (using Spark operators properties)
    - Bind join
- Benefits
  - Full power of data processing frameworks
    - No need for SQL-engines (like Hive)
  - Autonomy of the data stores
  - Eases query rewriting

Related Work

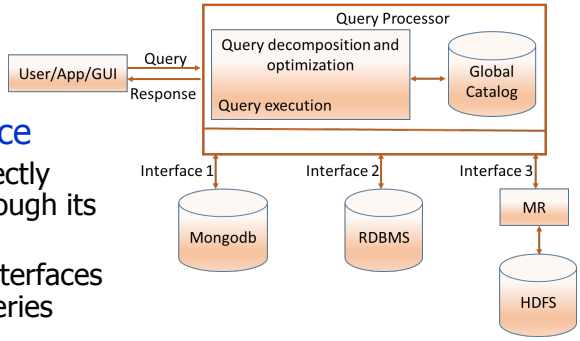
- Three kinds of multistores
  - Loosely-coupled
  - Tightly-coupled
  - Hybrid

Loosely-coupled Multistores



- Like multidatabase systems
  - Deal with autonomous data stores
  - One common interface to all data stores, translated to local interface
- Examples
  - ESTOCADA [Bugiotti, Inria, 2015]
  - BigIntegrator [Minpeng, Uppsala University,2011]

Tightly-coupled Multistores



- No common interface
  - Each data store directly accessed by QP through its local interface
  - QP uses the local interfaces to transform QL queries
- Optimization
  - Data movement across data stores
  - Use of materialized views or indexes
  - Examples
    - MISO [LeFevre , NEC Labs America, 2014]
    - Polybase [DeWitt, Microsoft Corporation, 2013]

Hybrid Multistores

- Support data source autonomy as in loosely-coupled systems
- Exploit the local data store interfaces as in tightly-coupled systems => optimization
- Examples
  - HadoopDB [Abouzeid, Yale and Brown Univ., 2009]
  - QOX [Simitsis, HP labs, 2012]

CloudMdsQL*

- Functional SQL-like query language for CoherentPaas's hybrid multistore
- Features
  - High expressivity
    - Allows the ad-hoc usage of user-defined Map/Filter/Reduce (MFR) operators, in combination with traditional SQL statements
  - Optimizability
    - By subquery rewriting and operation reordering, based on the properties of map, filter and reduce

*B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, J. Pereira. CloudMdsQL: Querying Heterogeneous Cloud Data Stores with a Common Language. *Distributed and Parallel Databases*, 43p, to appear, 2015.

Map/Filter/Reduce (MFR)

- A dataset is an abstration for a set of tuples such as a Resilient Distributed Dataset (Spark)
  - Consists of key-value tuples
  - Processed by MFR operations
- An MFR statement represents a sequence of Map/ Filter/Reduce operations on datasets
  - Example: count the words that contain the string 'cloud'

```
MAP(KEY,1).FILTER( KEY LIKE '%cloud%').REDUCE(SUM)
```

MFR Subquery Example

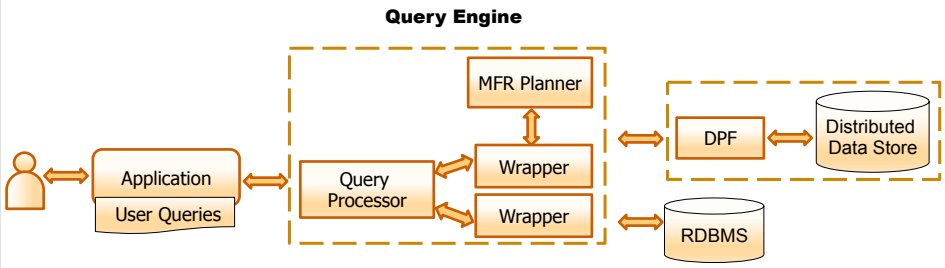
- Query: join two datasets retrieved from a relational database and a DPF

```
/* SQL subquery */
T1(title string, kw string)@rdb = ( SELECT title, kw FROM
tbl )

/* MFR subquery */
T2(word string, count int)@hdfs = {*
       SCAN(TEXT,'words.txt')
       .MAP(KEY,1)
       .REDUCE(SUM)
       .PROJECT(KEY,VALUE)  *}

/* Integration subquery*/
SELECT title, kw, count FROM T1 JOIN T2 ON T1.kw = T2.word
WHERE T1.kw LIKE '%cloud%'
```

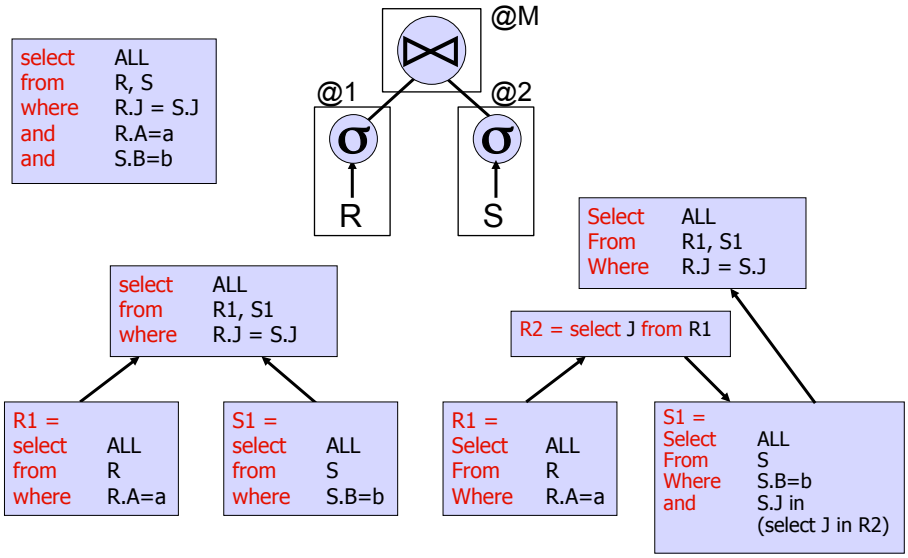

Query Engine



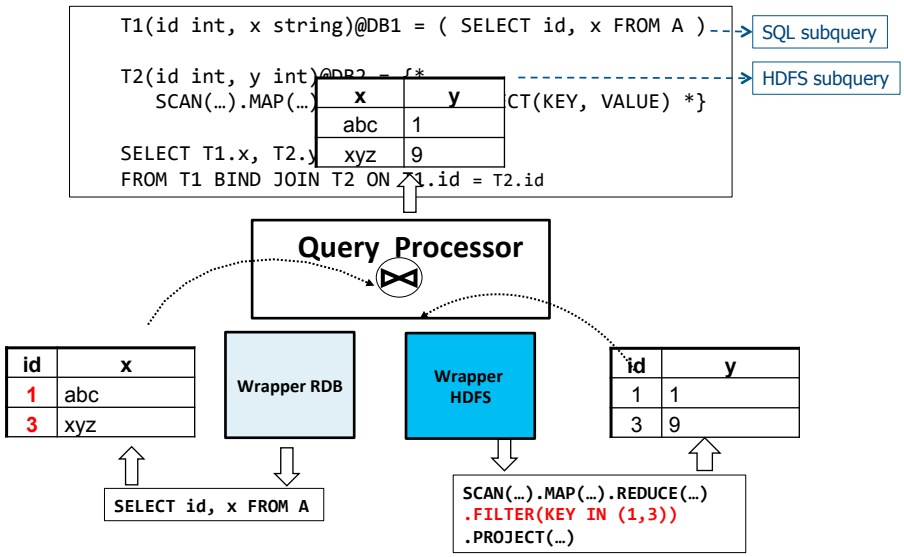
Query Rewriting

- Optimization techniques
  - Selection pushdown inside subqueries
    - Benefits from remote execution at the data store
  - Performing bind join
    - Reduces the communication cost between DPF and Query Engine
  - MFR operators reordering and rewriting
    - Reduces the amount of processed data

Optimization with Bindjoin



Subquery Rewriting: Bind Join

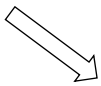


MFR Rewrite Rules

- Rules for reordering and rewriting MFR operators, based on their algebraic properties
- Rule #1

  Filter included before PROJECT

```
T1(a int, b int)@db1 ={* … .PROJECT (KEY, VALUE[0]) *}
SELECT a, b FROM T1 WHERE a > b
```



```
T1(a int, b int)@db1 ={* … .
FILTER(KEY>VALUE[0]).PROJECT(KEY,VALUE[0])*
}
SELECT a, b FROM T1
```

MFR Rewrite Rules

- Rule #2

  REDUCE(<transformation>).FILTER(<predicate>) =
  FILTER(<predicate>).REDUCE(<transformation>)

- Rule #3

  MAP(<expr_list>).FILTER(<predicate1>) =
  FILTER(<predicate2>).MAP(<expr_list>)

Map/Filter/Reduce => Spark

- We need to translate MFR operators to Spark operators
  - map
  - flatMap
  - reduceByKey
  - aggregateByKey
  - filter

Experimental Setup

1. Programming languages
   - Java for Query Engine
   - Python for MFR-Spark rewriting
2. RDBMS
   - PostgreSQL
3. Distributed file system
   - HDFS
4. Data processing framework
   - Spark
5. Cluster from Grid5000 platform
   - 1 node for Query Engine, 1 node for PostgreSQL, and *n* nodes for HDFS

Validation

- Reveal the steps the query engine take to process query
  - Use selection pushdown an bind join techniques
  - Rewrite the MFR subquery
- Evaluation of multistore query optimization techniques using our query engine

Validation

- We generated sample datasets in the context of the multistore query example

Publications

id	author	Title
1	Ricardo	Snapshot Isolation in Cloud DBs
3	Patrick	Principles of Distributed Cloud DBs
7	Larri	Graph DBs

Publications_keywords

pub-id	keyword
1	transactions
1	cloud
3	cloud
3	storage
7	graph
7	NoSQL

Posts

2014-11-10, alice, storage, cloud
2014-11-10, bob, cloud, virtual, app
2014-11-10, alice, cloud

Example: HDFS/RDBMS Query

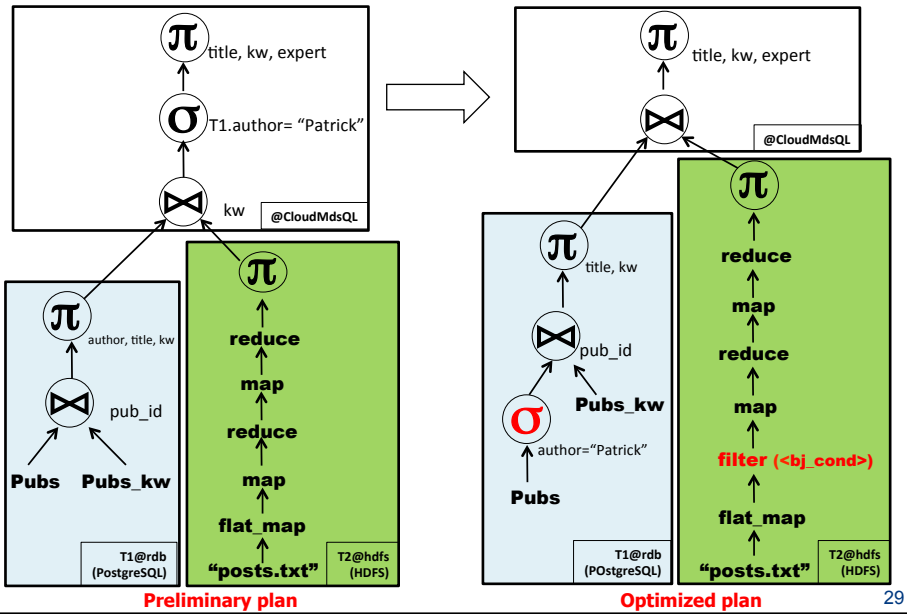
- Query: find appropriate reviewers for publications of a certain author, by considering each publication's keywords and the experts who have mentioned them most frequently on the scientific forum

```
-- SQL subquery
T1(autor, title, kw)@rdb = ( SELECT author, title, keyword
                             FROM pubs JOIN pubs_kw ON id = pub_id )

-- subquery in MFR notation
T2(kw, expert)@hdfs = {*
  SCAN(TEXT, 'posts.txt', ',')
     .FLAT_MAP( lambda data: (product(data[2:], [data[1]])))
     .MAP( TUPLE, 1 )
     .REDUCE( SUM )
     .MAP( KEY[0], (KEY[1], VALUE) )
     .REDUCE( lambda a, b: b if b[1] > a[1] else a )
     .PROJECT( KEY, VALUE[0] )
*}

--Integration subquery
SELECT T1.title, T1.kw, T2.expert
FROM T1, T2
WHERE T1.kw = T2.kw AND T1.author = 'Patrick'
```

Query Processing (1)



Query Processing (2)

Stage of optimization

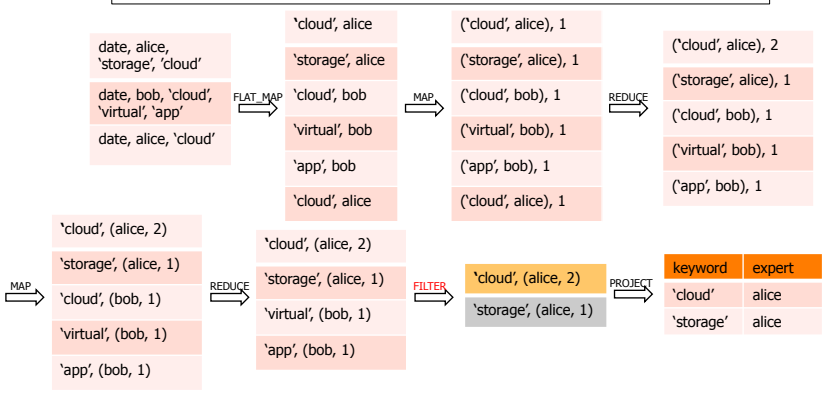
```
-- SQL subquery
T1(autor, title, kw)@rdb = ( SELECT author, title, keyword
                               FROM pubs JOIN pubs_kw ON id = pub_id
                               WHERE author = ’Patrick’)

-- subquery in MFR notation
T2(kw, expert)@hdfs = {*
  SCAN(TEXT, 'posts.txt', ‘,')
     .FLAT_MAP( lambda data: (product(data[2:], [data[1]]))
     .MAP( TUPLE, 1 )
     .REDUCE( SUM )
     .MAP( KEY[0], (KEY[1], VALUE) )
     .REDUCE( lambda a, b: b if b[1] > a[1] else a )
     .FILTER ( KEY IN (‘cloud’, ‘storage’))
     .PROJECT( KEY, VALUE[0] )
*}

--Integration subquery
SELECT T1.title, T1.kw, T2.expert
FROM T1, T2
WHERE T1.kw = T2.kw AND T1.author = ’Patrick’
```

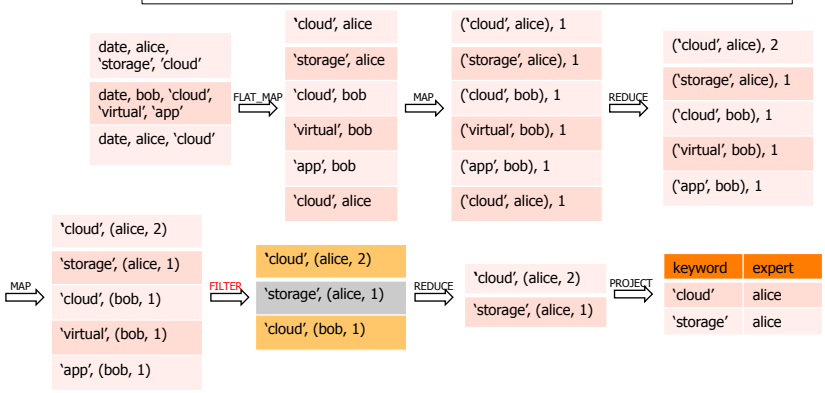
Selection Pushdown (1)

```
T2(kw, expert)@hdfs = {*
  SCAN(TEXT, 'posts.txt', ‘,')
     .FLAT_MAP( lambda data: (product(data[2:], [data[1]]))
     .MAP( TUPLE, 1 )
     .REDUCE( SUM )
     .MAP( KEY[0], (KEY[1], VALUE) )
     .REDUCE( lambda a, b: b if b[1] > a[1] else a )
     .FILTER( KEY IN (‘cloud’,’storage’ )
     .PROJECT( KEY, VALUE[0] )*}
```



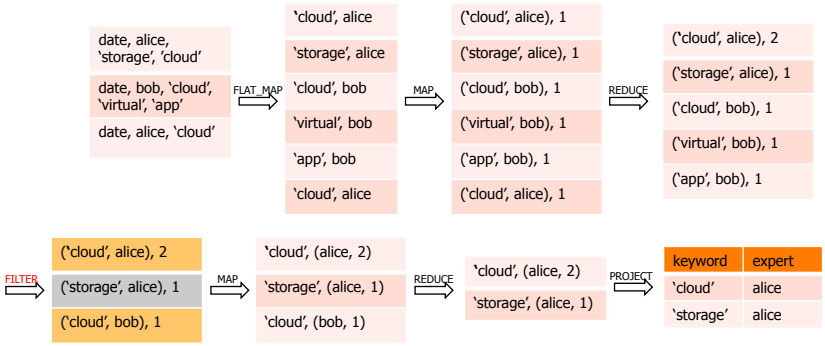
Selection Pushdown (2)

```
T2(kw, expert)@hdfs = {*
  SCAN(TEXT, 'posts.txt', ‘,')
     .FLAT_MAP( lambda data: (product(data[2:], [data[1]]))
     .MAP( TUPLE, 1 )
     .REDUCE( SUM )
     .MAP( KEY[0], (KEY[1], VALUE) )
     .FILTER( KEY IN (‘cloud’,’storage’ )
     .REDUCE( lambda a, b: b if b[1] > a[1] else a )
     .PROJECT( KEY, VALUE[0] )*}
```



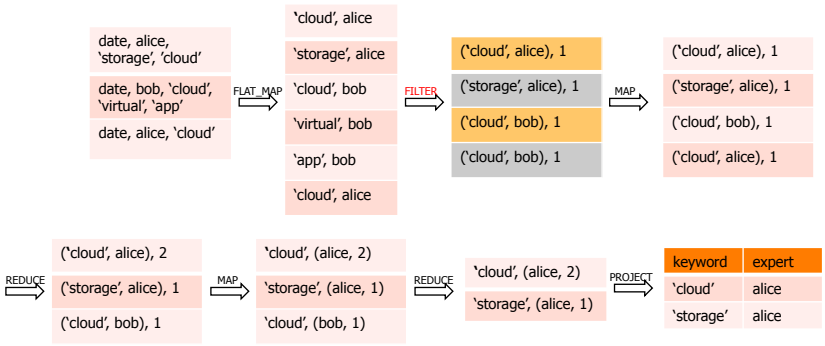
Selection Pushdown (3)

```
T2(kw, expert)@hdfs = {*
  SCAN(TEXT, 'posts.txt', ‘,')
     .FLAT_MAP( lambda data: (product(data[2:], [data[1]])))
     .MAP( TUPLE, 1 )
     .REDUCE( SUM )
     .FILTER( KEY[0] IN (‘cloud’,’storage’ )
     .MAP( KEY[0], (KEY[1], VALUE) )
     .REDUCE( lambda a, b: b if b[1] > a[1] else a )
     .PROJECT( KEY, VALUE[0] )*}
```



Selection Pushdown (4)

```
T2(kw, expert)@hdfs = {*
  SCAN(TEXT, 'posts.txt', ‘,')
     .FLAT_MAP( lambda data: (product(data[2:], [data[1]])))
     .FILTER( KEY[0] IN (‘cloud’,’storage’ )
     .MAP( TUPLE, 1 )
     .REDUCE( SUM )
     .MAP( KEY[0], (KEY[1], VALUE) )
     .REDUCE( lambda a, b: b if b[1] > a[1] else a )
     .PROJECT( KEY, VALUE[0] )*}
```



Rewriting to Spark

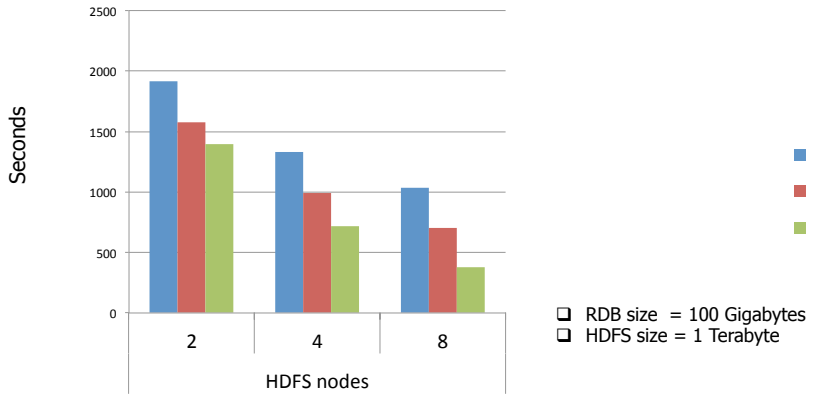
```
T2(kw, expert)@DB2 = {* SCAN(TEXT, 'posts.txt', ',')
.FLAT_MAP( lambda data: product(data[2:], [data[1]])
.FILTER(TUPLE [0] IN  'cloud' , 'storage'))
.MAP( TUPLE, 1)
.REDUCE(SUM)
.MAP( KEY[0], (KEY[1], VALUE) )
.REDUCE( lambda a, b: b if b[1] > a[1] else a )
.PROJECT( KEY, VALUE[0])*}
```

MFR code

Spark Code

```
T2(kw, expert)@DB2 = {* sc.textFile ('posts.txt', ',')
.map(lambda line: line.split(',')) \
.flatMap( lambda data: product(data[2:], [data[1]]) \
.filter( lambda tup: tup[0] in ['cloud‘, ‘storage’]) \
.map( lambda tup: (tup, 1)) \
.combineByKey( lambda a, b: a + b ) \
.reduceByKey( lambda a, b: a + b ) \
.map( lambda tup: (tup[0][0], (tup[0][1], tup[1]) ) )
\
.reduceByKey( lambda a, b: b if b[1] > a[1] else a ) \
.PROJECT( K, V[0], V[1] )*}
```


Performance Validation



- RDB size = 100 Gigabytes
- HDFS size = 1 Terabyte

Execution costs for 3 alternative plans and 3 different HDFS configurations

Conclusion

- CloudMdsQL: functional SQL-like language for hybrid multistore
  - Extended with a simple notation to specify the sequence of MFR operators for the data processing frameworks
- Query optimization techniques
  - Selection pushdown
  - Bind join
  - MFR operators reordering and rewriting
- Validation with PostgreSQL, Spark and HDFS