Apache MRQL (incubating): Advanced Query Processing for Complex, Large-Scale Data Analysis

Leonidas Fegaras

University of Texas at Arlington

http://mrql.incubator.apache.org/

04/12/2015

Outline

- Who am I?
- Motivation
- Design objectives
- Overview of MRQL
- Examples
- Demo
- Architecture
- Current work
- Future plans

About me

Leonidas Fegaras
fegaras@cse.uta.edu

- Associate Professor at UTA (Univ. of Texas at Arlington)
- A committer and PPMC member of Apache MRQL
- Interested in big data management:

  cloud computing, web data management, distributed computing, data stream processing, query processing and optimization
- Past projects:
  - HXQ: XQuery in Haskell
  - XStreamCast: query processing of streamed XML data
  - XQP: XQuery processing on P2P
  - XQPull: stream processing for XQuery
  - LDB: OODB query processing

History

Apache MRQL (incubating)

MRQL: a Map-Reduce Query Language

History:

- Fall 2010: started at UTA as an academic research project
- March 2013: enters Apache Incubation
- 3 releases under Apache so far: latest MRQL 0.9.4

Motivation

- MapReduce is not the only player in the Hadoop ecosystem any more
  - designed for batch processing
  - not well-suited for some big data workloads:
    real-time analytics, continuous queries, iterative algorithms, ...
- Alternatives:

  Spark, Flink, Hama, Giraph, ...
- New distributed stream processing engines:

  Spark Streaming, Flink Streaming, Storm, S4, Samza, ...

Motivation

- Designed to relieve application developers from the intricacies of big-data analytics and distributed computing
- Steep learning curve
- Hard to develop, optimize, and maintain non-trivial applications coded in a general-purpose programming language
- Hard to tell which one of these systems will prevail in the near future
    - applications coded in one of these paradigms may have to be rewritten as technologies evolve

Motivation

- ... or you can express your applications in a query language that is independent of the underlying distributed platform!

Motivation

- … or you can express your applications in a query language that is independent of the underlying distributed platform!
- Does it have to be SQL? We're noSQL after all!

Design objectives

- Wanted to develop a powerful and efficient query processing system for complex data analysis applications on big data
  - more powerful than existing query languages
  - able to capture most complex data analysis tasks <u>declaratively</u>
  - able to work on read-only, raw (in-situ), complex data
  - HDFS as the physical storage layer
  - platform-independent:
    - the same query can run on multiple platforms on the same cluster
    - allowing developers to experiment with various platforms effortlessly
  - efficient!

Design objectives

- We envision MRQL to be:
  - a common front-end for the multitude of distributed processing frameworks emerging in the Hadoop ecosystem
  - a tool for comparing these systems (functionality & performance)

Oh great! yet another SQL for map-reduce

- MRQL is **NOT** SQL!
- MRQL is an SQL-like query language for large-scale, distributed data analysis on a computer cluster
- Unlike SQL, MRQL supports
  - a richer data model (nested collections, trees, ...)
  - arbitrary query nesting
  - more powerful query constructs
  - user-defined types and functions
- no nulls, no outer-joins
- MRQL queries can run on multiple distributed processing platforms

  currently Apache Hadoop MapReduce, Hama, Spark, and Flink
- The MRQL syntax and semantics have been influenced by
  - modern database query languages (mostly, XQuery and ODMG OQL)
  - functional programming languages (sequence comprehensions, algebraic data types, type inference)

Language features

The MRQL query language:

- provides a rich type system that supports hierarchical data and nested collections uniformly
  - general algebraic datatypes (similar to Haskell)
    - JSON and XML are user-defined types
  - pattern matching over data constructions (similar to 'case' in Haskell)
  - local type inference (similar to Scala)
- allows nested queries at any level and at any place
  - no need for awkward nulls and outer-joins
- supports UDFs
  - provided that they don't have side effects
- allows to operate on the grouped data using queries
  - as is done in OQL and XQuery
  - improves SQL group-by/aggregation (which are too awkward)

Language features

The MRQL query language:

- supports custom aggregations/reductions using UDFs

  provided they have certain properties (associative & commutative)
- supports iteration declaratively

  to capture iterative algorithms, such as PageRank
- supports custom parsing and custom data fragmentation
- provides syntax-directed construction/deconstruction of data

  to capture domain-specific languages

How does MRQL compare to Hive?

	MRQL	Hive
metadata	none	stored in RDBMS
data	nested collections, trees, and custom complex types	relational
group-by	on arbitrary queries	not on subqueries
aggregation	arbitrary queries on grouped data	SQL aggregations
subqueries	arbitrary query nesting	limited subquery support
platforms	Hadoop, Hama, Spark, Flink	Hadoop, Tez, (Spark)
file formats	text, sequence, XML, JSON	text, sequence, ORC, RCFile
iteration	yes	no
streaming	yes	no

Simple example: matrix multiplication

A sparse matrix X is represented as a bag of (X_{ij}, i, j).

$$Z_{ij} = \sum_k X_{ik} * Y_{kj}$$

```
select ( sum(z), i, j )
  from (x,i,k) in X, (y,k,j) in Y, z = x*y
 group by i, j
```

An XML example

Group all persons according to their interests and the number of open auctions they watch. For each such group, return the number of persons in the group:

```
select ( cat, os, count(p) )
from p in XMARK,
     i in p.profile.interest
group by cat: i.@category,
         os: count(p.watches.@open_auctions)
```

Example: k-means clustering

Derive *k* clusters from a set of points P:

```
repeat centroids = ...
  step select < X: avg(s.X), Y: avg(s.Y) >
         from point in Points
        group by k: (select c from c in centroids
                         order by distance(point,c))[0]
```

Example: the PageRank algorithm

Simplified PageRank:

- A graph node is associated with a PageRank and its outgoing links:

  < id: 23, rank: 0.0, adjacent: { 10, 45, 35 } >

- Propagate the PageRank of a node to its outgoing links; each node gets a new PageRank by accumulating the propagated PageRanks from its incoming links:

```
repeat nodes = ...
  step select < id: m.id, rank: n.rank, adjacent: m.adjacent >
       from n in (select < id: key, rank: sum(c.rank) >
                    from c in ( select < id: a,
                                         rank: n.rank/count(n.adjacent) >
                                  from n in nodes,
                                       a in n.adjacent )
                   group by key: c.id ),
            m in nodes
       where n.id = m.id
```

The complete PageRank using map-reduce (MR)

```
graph = select ( key, n.to )
         from n in source(line,"graph.csv",...)
        group by key: n.id;                            preprocessing: 1 MR job

size = count(graph);

select ( x.id, x.rank )
from x in
    (repeat nodes = select < id: key, rank: 1.0/size, adjacent: al >
                     from (key,al) in graph            init step: 1 MR job
       step select (< id: m.id, rank: n.rank, adjacent: m.adjacent >,
                     abs((n.rank-m.rank)/m.rank) > 0.1)
             from n in (select < id: key, rank: 0.25/size+0.85*sum(c.rank) >
                         from c in ( select < id: a, rank: n.rank/count(n.adjacent) >
                                      from n in nodes, a in n.adjacent )
                        group by key: c.id),
                  m in nodes
            where n.id = m.id)                         repeat step: 1 MR job
order by x.rank desc;                                  postprocessing: 1 MR job
```

Demo

Demo link

Architecture

Query translation stages:

1. type inference
2. query translation and normalization
3. simplification
4. algebraic optimization
5. plan generation
6. plan optimization
7. compilation to Java code

The essence of distributed data processing

- distribute data to worker nodes (shuffling)
- perform computations on each data partition
- combine the results of these computations into one result

Algebraic operators

- Algebraic operations on bags:

groupBy	(X: $\{(\kappa,\alpha)\}$)	: $\{(\kappa,\{\alpha\})\}$
flatMap	(f: $\alpha \rightarrow \{\beta\}$, X: $\{\alpha\}$)	: $\{\beta\}$
reduce	($\oplus$: $(\alpha,\alpha) \rightarrow \alpha$, X: $\{\alpha\}$)	: α
union	(X: $\{\alpha\}$, Y: $\{\alpha\}$)	: $\{\alpha\}$

- Extra operation (join):
  coGroup (X: $\{(\kappa,\alpha)\}$, Y: $\{(\kappa,\beta)\}$) : $\{(\kappa,\{\alpha\},\{\beta\})\}$
- map-reduce = flatMap $\circ$ groupBy $\circ$ flatMap
- List operations: orderBy, append
- Iteration: repeat (f: $\{\alpha\} \rightarrow \{\alpha\}$, X: $\{\alpha\}$) : $\{\alpha\}$

Query optimization

The query optimizer:

- uses a cost-based optimization framework to map algebraic terms to efficient workflows of physical operations
- handles dependent joins (used for nested collections)
- unnest deeply nested queries and converts them to join plans

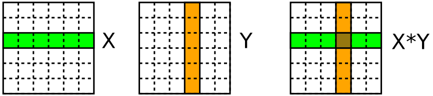
Physical operations

Case study: Queries similar to matrix multiplication:

```
select ( sum(z), i, j )
  from (x,i,k) in X, (y,k,j) in Y,
       z = x*y
 group by i, j
```

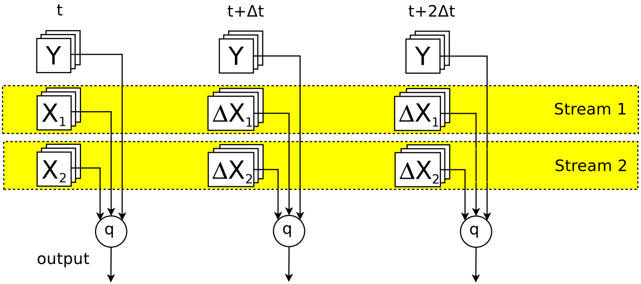
```
select h( k, reduce(acc,z) )
  from x in X, y in Y, z = f(x,y)
 where jx(x) = jy(y)
 group by k: ( gx(x), gy(y) )
```

- GroupByJoin (Valiant's algorithm): distribute the data to workers in the form of a grid $n \times n$ of partitions
  - each partition contains only those rows from X and those columns from Y needed to compute a single partition of the resulting matrix
  - X and Y values are replicated n times
  - each worker uses $(|X| * |Y|)/n^2$ memory



Current work: distributed stream processing

- Support for continuous queries over multiple streams of data
- Data come in incremental batches ΔX
- Batch streaming based on sliding windows



Query $q(X_1, X_2; Y)$ over one invariant and two streaming data sources

Current work: distributed stream processing

```
select (k,avg(p.Y))
  from p in stream(binary,"points")
 group by k: p.X
```

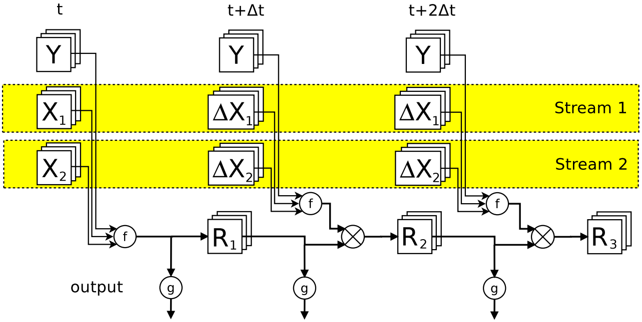
- Currently, works on Spark Streaming
- Soon, on Flink Streaming

Next step: incremental query processing

- *Problem:* translate any batch program (eg, PageRank) to an incremental program automatically
- *Solution:* Break the query $q(X_1, X_2; Y) = g(f(X_1, X_2; Y))$ such as:

$$f(X_1 \uplus \Delta X_1, X_2 \uplus \Delta X_2; Y) = f(X_1, X_2; Y) \otimes f(\Delta X_1, \Delta X_2; Y)$$

- Requires program analysis & transformation



Summary

- Slides are available at the MRQL wiki page:
  `http://wiki.apache.org/mrql/`
- We are looking for new developers to work on open tasks:
  - add support for more distributed/streaming platforms
    - Storm
  - support more input formats, including key-value stores
  - implement incremental query processing
  - specify more data analysis algorithms
  - benchmarking
- Are you developing a distributed processing platform in need for a query language?

  talk to us!