RDF Graph Pattern Matching on Map-Reduce

Team name

N N N

Member

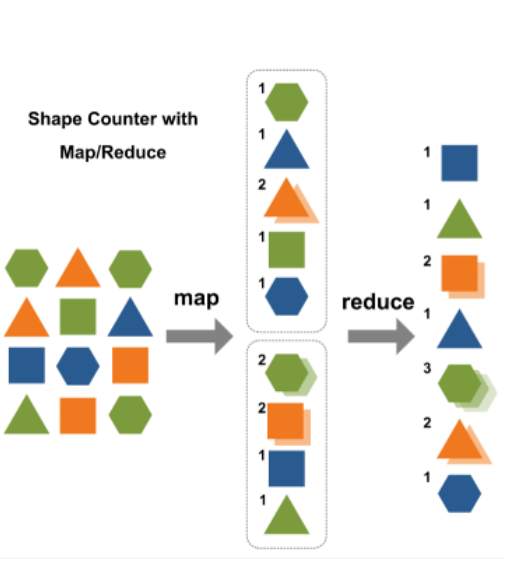
Prof	O O O
TA	O O O
Teammate	O O O
	O O O
	O O O

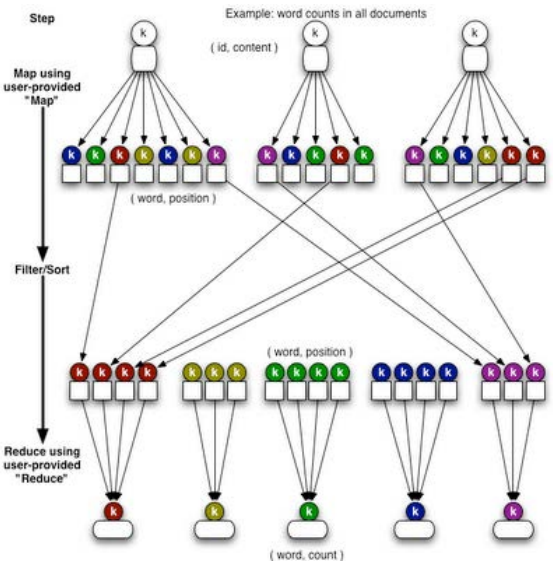
Abstraction

- Using NTGA(Nested TripleGroup Algebra), improve Map-Reduce phase of Pig Latin. And Design and implement RDF Graph Pattern Matching API on Apache Hadoop Framework.

Introduction

- MapReduce is a programming model for processing large data sets. MapReduce is typically used in distributed computing on clusters of computers.

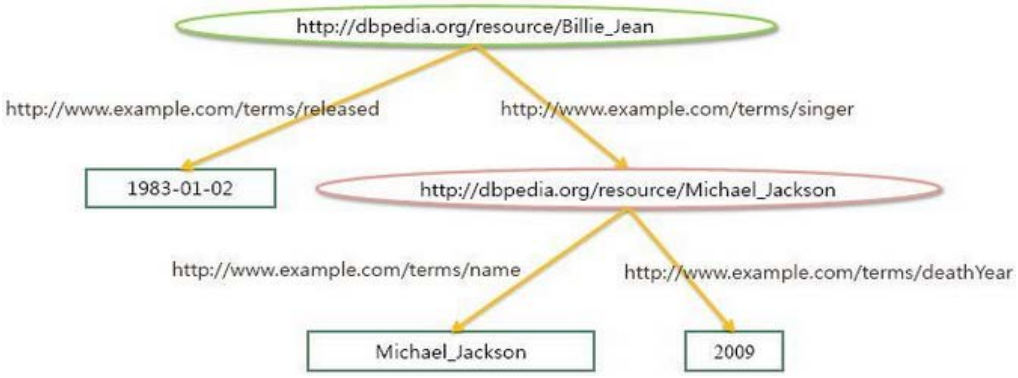




- RDF is Resource Description Framework that is a W3C standard model for data

interchange on the Web.

- RDF Graph Pattern Matching make it possible analytic processing from semantic web data by RDF triple group queries like SQL from-where statement.



- We are going to design and implement RDF Graph Pattern Matching API on Apache Hadoop framework.

Existing Algorithm

- Existing Algorithm using SPARQL and Pig Latin (based on Hadoop)

```
SELECT  ?vlabel ?hpage ?prod
        ?valTo ?price ?rev ?rat1
WHERE { ?v homepage ?hpage .
        ?v label ?vlabel .                    } SJ1
  J1    ?v country ?vcountry .
        ?o price ?price.
        ?o vendor ?v .
        ?o delivDays ?delDays                 } SJ2
        ?o validTo ?valTo .
  J2    ?o product ?prod .
        ?r reviewFor ?prod .
        ?r reviewer ?rev .                    } SJ3
        ?r rating1 ?rat1 .
FILTER ( ? delivDays < 3  &&
        ?vcountry == "US")
```

(a)

```
A= LOAD 'input.nt' using PigStorage(' ') as (S, P, O)
SPLIT A into
  country if P eq 'country' and O eq 'US',
  vlabel if P eq 'label', price if P eq 'price',
  hpage if P eq 'homepage',
  delDays if P eq 'delivDays' and O < 3,
  valTo if P eq 'validTo',  vendor if P eq 'vendor',
  prod if P eq 'product', revFor if P eq 'reviewFor',
  rev if P eq 'reviewer', rat1 if P eq 'rating1';
SJ1 = JOIN vlabel by S, hpage by S, country by S;
SJ2 = JOIN price by S, delDays by S, valTo by S,
          prod by S, vendor by S;
SJ3 = JOIN revFor by S, rev by S, rat1 by S;
J1  = JOIN SJ1 by $0, SJ2 by $2;
J2  = JOIN J1 by $20, SJ3 by $2;
STORE J2 into '/graphPatternResults';
```

(b)

Figure 1 **Example pattern matching query in (a) SPARQL (b) Pig Latin (VP approach)**

- Existing algorithm based on Hadoop realize MapReduce process by using recursive JOIN

function. And it merges and groups each JOIN function by using FILTER and SPLIT functions. But each JOIN function makes I/O overhead, cause of many JOIN function and FILTER combination, it must make many overhead and resource waste.

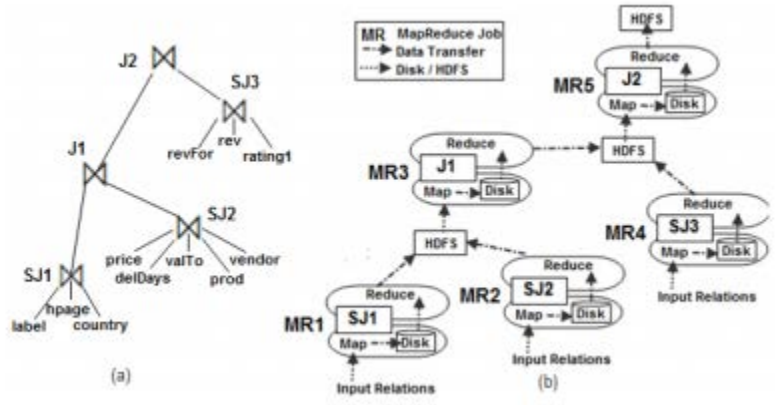


Figure 2 **Pattern Matching using VP approach (a) Query plan (b) Map-Reduce execution flow**

Nested Triple Group Algebra

tg = {(&V1, *label*, vendor1), (&V1, *country*, US), (&V1, *homepage*, www.vendors.org/V1)}	**ntg** = {(&Offer1, *price*, 108), (&Offer1, *validTo*, 2012/12/31), (&Offer1, *vendor*, {(&V1, *label*, vendor1), (&V1, *country*, US), (&V1, *homepage*, www.vendors.org/V1)}), (&Offer1, *product*, &P1)}
(a)	(b)

- NTG (Nested TripleGroup) consists of a root TripleGroup and one or more child TripleGroups. An example of a nested TripleGroup is shown in figure (b).

ntg = {(&Offer1, *price*, 108),
(&Offer1, *validTo*, 2012/12/31),
(&Offer1, *vendor*, {(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, www.vendors.org/V1)})
(&Offer1, *product*, &P1)}

unnest

untg = {(&Offer1, *price*, 108),
(&Offer1, *validTo*, 2012/12/31),
(&Offer1, *vendor*, &V1),
(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, www.vendors.org/V1),
(&Offer1, *product*, &P1)}

tg = {(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, www.vendors.org/V1)}

flatten

nt = (&V1, *label*, vendor1, &V1, *country*, US, &V1, *homepage*, www.vendors.org/V1)
t1 *t2* *t3*

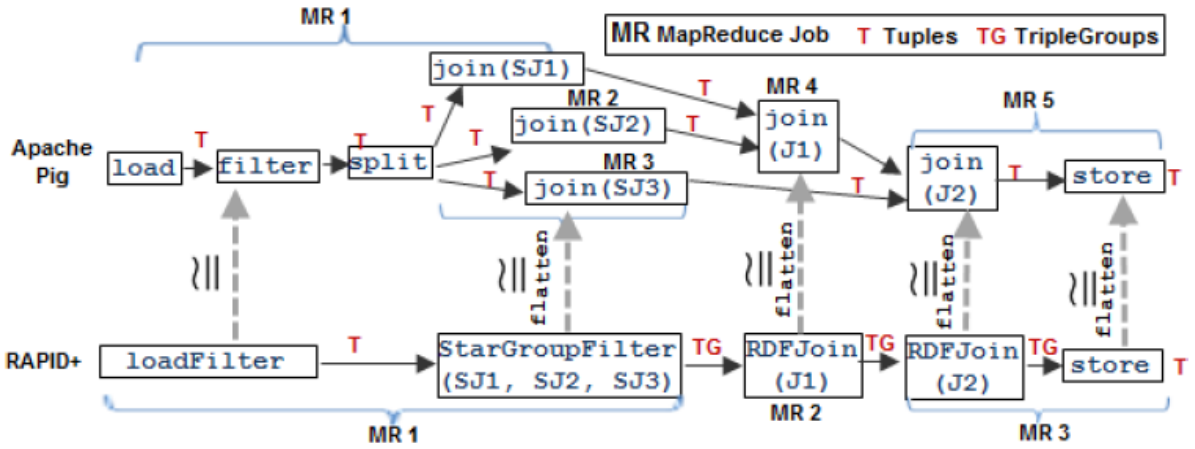
- **Nested TripleGroup** can be `unnested` and `flattened`. TripleGroup is changed into n-tuple after flatten operator. We can see that the information content in both formats is equivalent. We refer to this kind of equivalence as content equivalence. Consequently, computing query results in terms of TripleGroups is lossless in terms of information.

- **`LoadFilter`** : Pig load and filter(check if the TripleGroups satisfy the given filter condition. Similar to WHERE clause of SQL) operators are coalesced into a loadFilter operator to minimize costs of repeated data handling.

- **`StarGroupFilter`** : used for structure-based filtering. star-joins using Pig's GROUP BY operator is coalesced with the StarGroupFilter.

- **`RDFJoin`** : computes the join between a TripleGroup tgx in equivalence class TGx with a TripleGroup tgy in equivalence class TGy based on the given triple patterns.

Suggestion



- Above figure shows the equivalence between NTGA and relational algebra operator.

- In Above Figure, Apache Pig Latin has 2N-1 of Map-Reduce processes. On the other hand, the one using NTGA has N times of Map-Reduce processes.

Back-end

- **Data Types**
  - A relation is a bag.
  - A bag is a collection of tuples.
  - A tuple is an ordered set of fields.
  - A field is a piece of data.

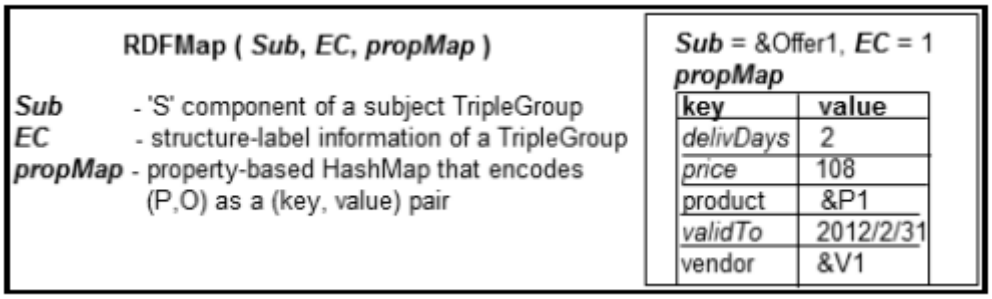
- **User Defined Function**
  - To extend Pig Latin and Hadoop's functionality and specify custom processing
  - Can be written in Java, Python, JavaScript and Ruby, but Java is the most extensive.
  - Supports to extend eval, aggregate, algebraic, accumulator, filter, load, store, schemas, function overloading, and so on.

Data Structure

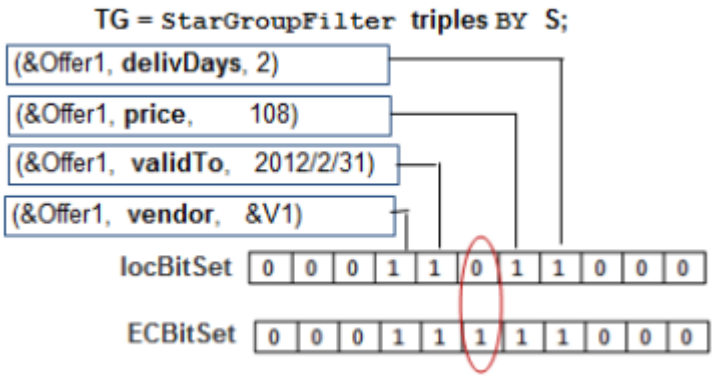
- **RDFMap**
  - Pig Latin's Data bag is inefficient for NTGA operators because it is an iterative data structure implemented as an array list of tuples.
  - RDFMap is a specialized data structure targeted at efficient implementation of NTGA operators.

```
public RDFMap(Tuple subject, int ec, Map propMap)
```

  - TripleGroup := {(s, p1, o1), (s, p2, o2), ...}
  - RDFMap := (s, ec, {p1: o1, p2: o2, ...})



User Defined Function

StarGroupFilter	used for structure-based filtering. star-joins using Pig's GROUP BY operator is coalesced with the StarGroupFilter.
example	
code	


```java
public Tuple map(Tuple triple) {
    Tuple tuple = TupleFactory.getInstance().newTuple(2);
    tuple.set(0, triple.get(0));
    tuple.set(1, triple);
    return tuple;
}

public RDFMap reduce(K key, Iterator<Tuple> tuples) {
    for (Tuple tuple : tuples) {
        subject = tuple.get(0);
        ec = findEC(tuple.get(1));
        locBitset.set(tuple.get(1));
        propMap.put(tuple.get(1), tuple.get(2));
    }
    ecBitset = getECBitset(ec);
    if (!locBitset.equals(ecBitset)) {
        return null;
    } else {
        return new RDFMap(subject, ec, propMap);
    }
}
```

RDFJoin	computes the join between a TripleGroup tgx in equivalence class TGx with a TripleGroup tgy in equivalence class TGy based on the given triple patterns.
example	Object-Subject Join on RDFMaps J1 = RDFJoin TG ON (1:'vendor'; 0:*);

Sub = &Offer1.&V1, ***EC*** = 1.0
propMap

key	value
delivDays	2
price	108
product	&P1
validTo	2012/2/31
label	vendor1
country	US
homepage	www.vendors.org/V1

code

```
public Tuple map(RDFMap rdfMap) {
    Tuple tuple = TupleFactory.getInstance().newTuple(2);
    if (joinKey == STAR_JOIN) {
        tuple.set(0, rdfMap.getSubject());
    } else {
        tuple.set(0, rdfMap.getPropMap().get(joinKey));
    }
    tuple.set(1, rdfMap);
    return tuple;
}

public List<RDFMap> reduce(K key, Iterator<RDFMap> rdfMaps) {
    for (RDFMap rdfMap : rdfMaps) {
        if (rdfMap.getEC() == EC1) {
            outerList.add(rdfMap);
        } else if (rdfMap.getEC() == EC2) {
            innerList.add(rdfMap);
        }
    }
    for (RDFMap outer : outerList) {
        for (RDFMap inner : innerList) {
            RDFMap rdfMap = joinRDFMap(outer, inner);
            resultList.add(rdfMap);
        }
    }
    return resultList;
}
```

Front-end

- **Eclipse Plug-in**
  - To improve existing PigPen plug-in
  - Supports to use RDFMap and NTGA operators
  - Executes the extended Pig Latin on Hadoop and shows the result of analysis.
  - Syntax highlighting for NTGA Algebra such as StarGroupFilter, RDFJoin

```
A=LOAD 'input.nt' using PigStorage('') as(S, P, O)
split  A into
    country if P eq 'country' and O eq 'US',
    vlabel if P eq 'label', price if P eq 'price',
    hpage if P eq 'homepage',
    delDays if P eq 'delivDays' and O<3,
    valTo if P eq 'validTo', vendor if P eq 'vendor',
    prod if P eq 'product', revFor if P eq 'reviewFor',
    rev if P eq 'reviewer', rat1 if P eq 'rating1';
SJ1=join vlable by S, hpage by S, country by S;
SJ2=join price by S, delDays by S, valto by S,
        prod by S, vendor by S;
SJ3=join revFor by S, rev by S, rat1 by S;
J1=join SJ1 by $0, SJ2 by $2;
J2=join J1 by $0, SJ3 by $2;
store J2 into '/graphPatternResults';
```

Goals

- By using Apache Hadoop, having high utilization at real world, prove the less overhead method RDF graph pattern matching. And open our API, language specification, Eclipse Plug-in to contribute FOSS community.

1. **Design extend language specification & Realize API layer**

   - Design extended language specification focused on RDF graph pattern matching. To support it, extend existing Apache Hadoop's internal data structure & process to realize semantic analyzing API layer. It'll be supported to open source, so it can be applied to many project needs mass storage semantic analysis.

2. **Eclipse Plug-in development**

   - Develop the efficient GUI supporting developed API layer and extend language specification. Eclipse is very general tool in Java side development, having wide application and extension.

3. **Verify & Result discussion**

   - Compare existing SPAQL based non-MapReduce analyze method with MapReduce

process efficiency. And simulate how efficient new extend language can reduce JOIN cost comparing with Pig Latin's simple expression by using big RDF data sample.

Progress

- Finished analysis of paper and RDF graph pattern matching.
- Started to development Eclipse Plug-in.
- Analyzing and implementing on Pig Latin and Hadoop framework.

Milestone

	Oct					Nov				Dec
Week	5	6	7	8	9	10	11	12	13	14(exhibition)
Analysis of Paper And RDF graph pattern matching										
Analysis of Apache Hadoop framework and Pig Latin										
Developing Eclipse Plug-in										
Language Specification Design And Implementation API Layer										
Test And Result Analysis										

O O : O O O : O O O :