Efficient Processing of RDF Graph Pattern Matching on MapReduce Platforms

Padmashree Ravindra, Seokyong Hong,
HyeongSik Kim, **Kemafor Anyanwu**

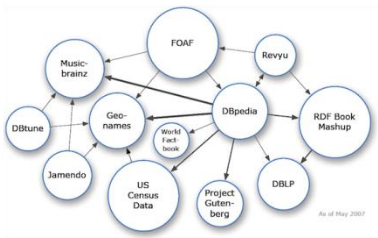
COUL – Semantic COmpUting research Lab

Computer Science
NC STATE UNIVERSITY

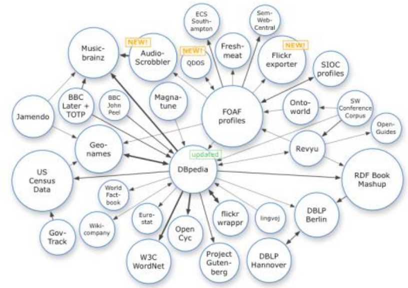
Outline

- ✓ Background
  - Semantic Web (RDF, SPARQL)
  - Join Processing in MapReduce framework
  - RDF Graph Pattern Matching in Apache Pig
- ✓ Challenges
- ✓ Approach
  - Algebraic Optimization – TripleGroup based Processing
  - Dynamic Optimization – Information Passing
- ✓ Evaluation
- ✓ Related Work
- ✓ Conclusion and Future Work

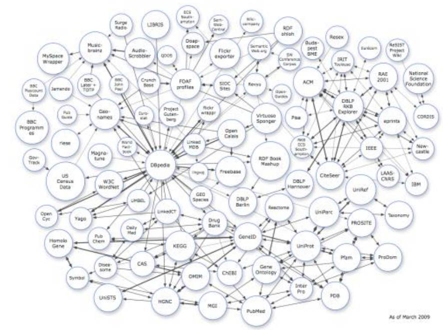
Linked Data and the Semantic Web



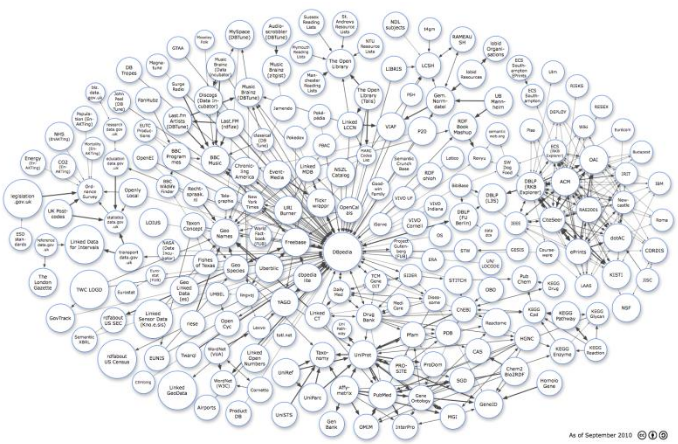
May 2007 - # of datasets: 12



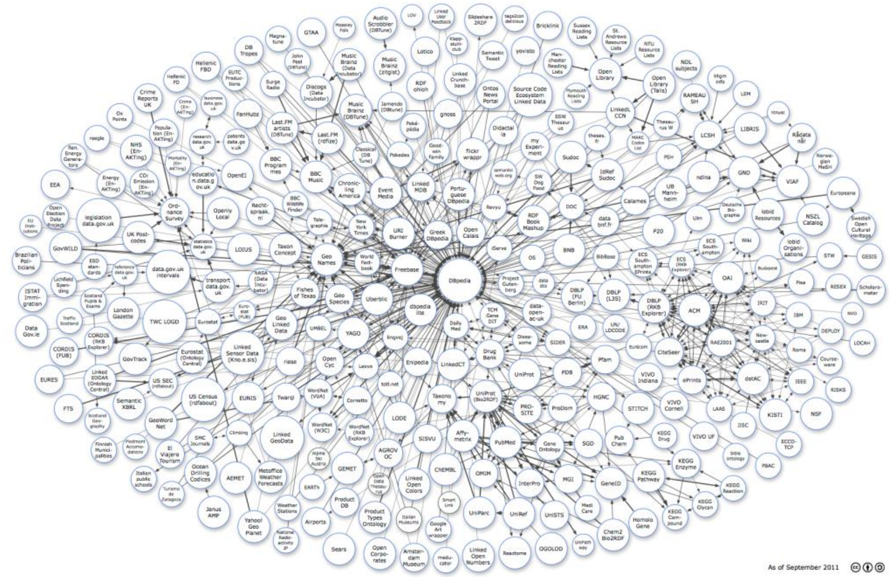
Feb 2008 - # of datasets: 32



March 2009 - # of datasets: 93



Sep 2010 - # of datasets: 203



Sep 2011 - # of datasets:**295**

Growing #RDF triples: currently **31 billion**

Computer Science
NC STATE UNIVERSITY

Example RDF Data and SPARQL Query

Statements (triples)

Sub	Prop	Obj
&V1	*type*	VENDOR
&V1	*country*	US
&V1	*homepage*	www.vendor...
&Offer1	*vendor*	&V1
&Offer1	*price*	108
	….	

Data: BSBM benchmark data describing *Vendors* and their *Offers*

Query: Retrieve the homepage of US based Vendors

SELECT ?hpage
WHERE {?s *type* VENDOR .
?s *country* ?vcountry .
?s *homepage* ?hpage .}
FILTER (?vcountry = "US");

Implicit Join based on common variables

&V1	*type*	VENDOR	&V1	*country*	US	&V1	*homepage*	www.vendor...

⋈ (between the *type* and *country* triples) ⋈ (between the *country* and *homepage* triples)

#Required Joins = 2

Several joins for more complex pattern matching tasks

Computer Science
NC STATE UNIVERSITY

Our Direction

- Need: Scalable and cost-effective processing techniques to deal with growing amount of Semantic Web data
- Unlikely to achieve good scalability without parallelization
- *MapReduce* platforms offer scalability in an easy-to-use and cost effective manner
- BUT expensive for multi-join queries typical of Semantic Web processing e.g. SPARQL query processing

Computer Science
NC STATE UNIVERSITY

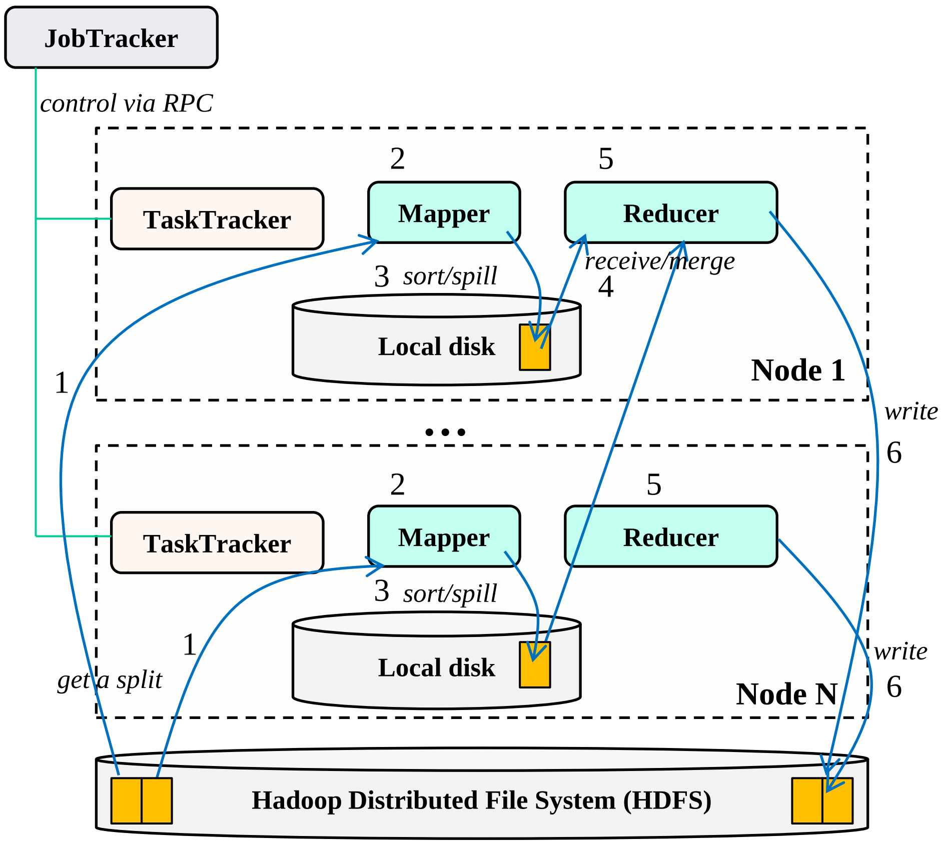
Basics: MapReduce

- Large scale processing of data on a cluster of commodity grade machines
- Users encode task as *map* / *reduce* functions, which are executed in parallel across the cluster
- Apache Hadoop* – open-source implementation
- Key Terms
  - ✓ Hadoop Distributed File System (HDFS)
  - ✓ Slave nodes / Task Tracker – Mappers (Reducers) execute the *map* (*reduce*) function
  - ✓ Master node / Job Tracker – manages and assigns tasks to Mappers / Reducers

Computer Science
NC STATE UNIVERSITY

* http://hadoop.apache.org/

1 MR cycle in Hadoop

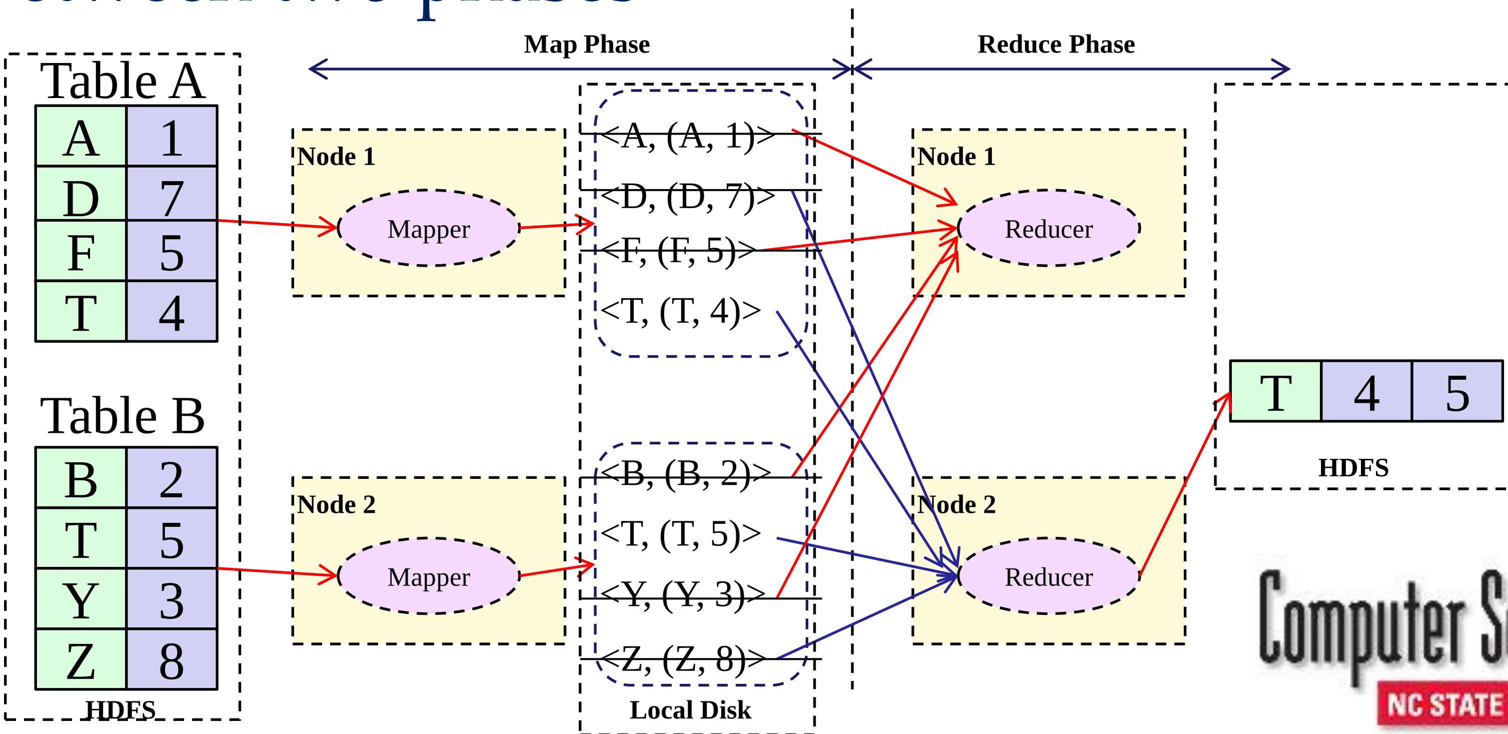


1. Mappers load splits **(I/O)**
2. Mappers process splits by executing ***map()***
3. Mappers sort/spill **(CPU/I/O)** intermediate <*key*, *value*>
4. Reducers retrieve intermediate <*key*, *value*> from mappers **(communication, I/O)**
5. Reducers process data by executing ***reduce()***
6. Reducers store resulting <*key*, *value*> tuples to HDFS **(I/O)**

Computer Science
NC STATE UNIVERSITY

Join Processing on Hadoop

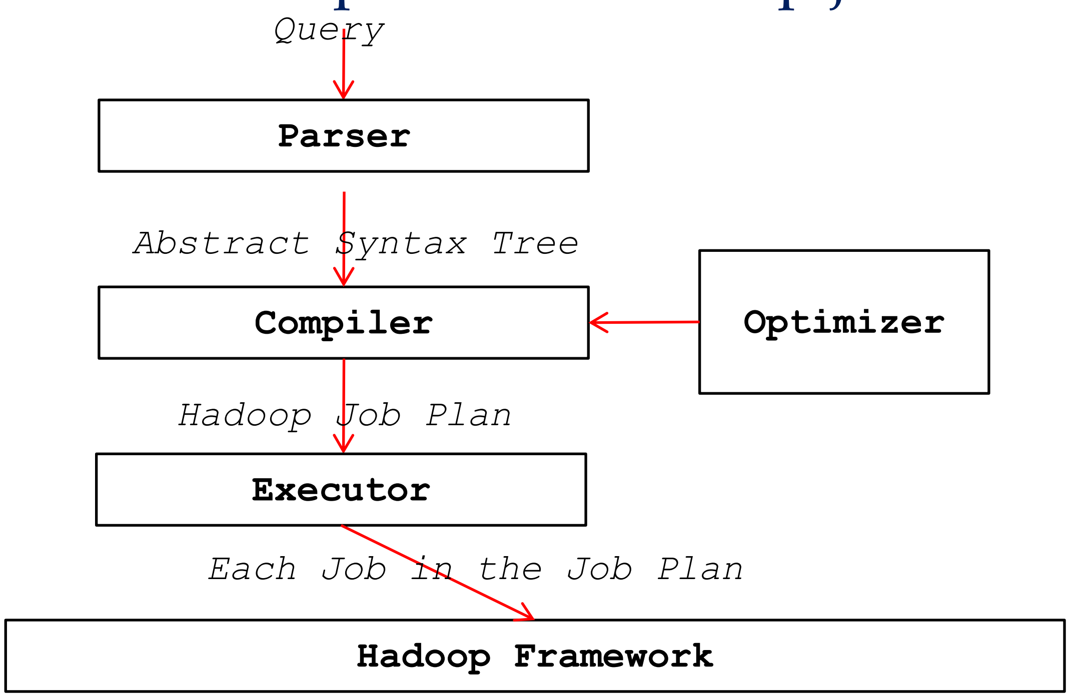
- ➢ Standard Repartitioning Join (*reduce-side*)
  - ✓ *map()*: "tag" tuple based on join key
  - ✓ *reduce()*: collect tuples with same "tag" and join relevant tuples
- ➢ Problem: all tuples in both relations (*should be distinguishable*) need to be sorted and transferred between two phases



Computer Science
NC STATE UNIVERSITY

Complex Query Processing on Hadoop

- Low level implementation a burden for users
- Pig/Hive: Allow expression of tasks using high-level query primitives
  - ✓ usability, code reuse, automatic optimization
  - ✓ Support for *relational-style* ops – `Join`, `Group By`
  - ✓ Operators compile into Hadoop jobs

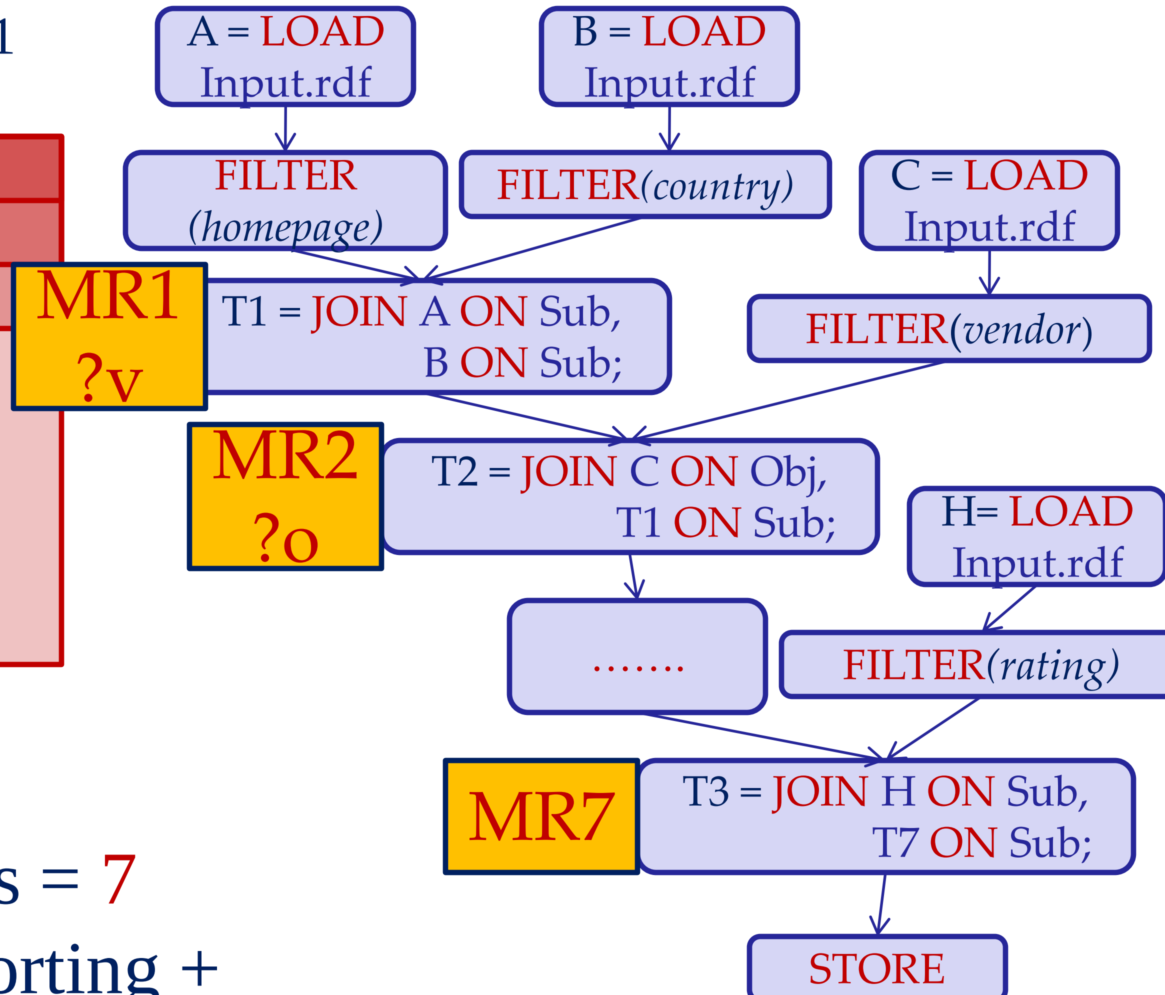


Computer Science
NC STATE UNIVERSITY

RDF Data Processing on Hadoop

SELECT ?hpage ?price ?rat1
WHERE

{?v	*homepage*	?hpage .
?v	*country*	?vcountry .
?o	*vendor*	?v .
?o	*price*	?price .
?o	*product*	?prod .
?r	*revFor*	?prod .
?r	*reviewer*	?rev .
?r	*rating*	?rat1 .}



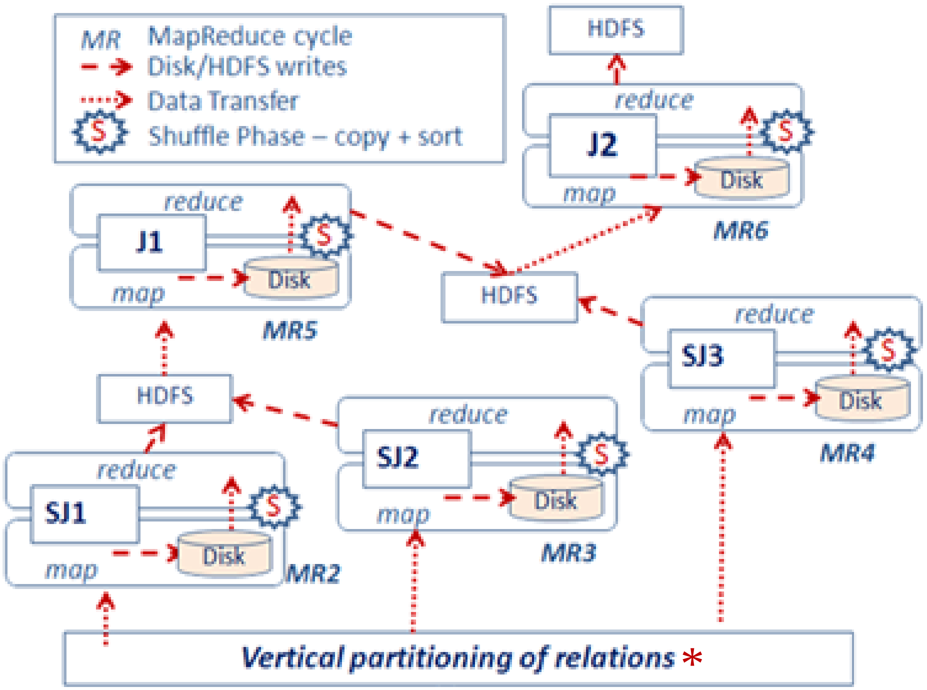
Execution Plan in Pig

#MR cycles = #Joins = 7

→(Data loading + sorting + transfer + materialization costs) * 7

Expensive!!!

Star-joins via *m-way* JOIN



SELECT ?hpage, ?price, ?rat1
WHERE { ?v *homepage* ?hpage .
?v *country* ?vcou.. .
?o *vendor* ?v .
?o *price* ?price .
?o *product* ?prod .
?r *revFor* ?prod .
?r *reviewer* ?rev .
?r *rating1* ?rat1 .}

SJ1: ?v homepage, ?v country
SJ2: ?o vendor, ?o price, ?o product
SJ3: ?r revFor, ?r reviewer, ?r rating1
J1 (obj-sub): ?v country, ?o vendor
J2 (obj-obj): ?o product, ?r revFor

#MR cycles reduced from 7 to 5

Can we do better???

**vertical-partitioning* of triple relation based on properties to avoid self-joins on large relations

Computer Science
NC STATE UNIVERSITY

How to reduce these costs?

- ✓Goal1: Minimize the length of MapReduce execution workflows
  - → Reduce #iterations for disk I/O, communication and sorting
- ✓Goal2: Minimize size of intermediate data
  - → Reduce the #tuples sorted and transferred between the nodes

Goal1: Minimizing #MR cycles

- Concurrent processing of star-joins can further reduce the required #MR cycles
- Challenge: Requires support for *inter-operator* parallelism in Hadoop
  - → Changes to scheduler + complex partitioning scheme

What are we proposing?

- An algebra (*Nested TripleGroup Algebra - NTGA*) for more efficient processing of RDF graph patterns based on a nested *Tripl*
  - ✓ Do ll the time!!!

Star-joins SJ1, SJ2, SJ3 require ONLY 1 MR cycle!!!

Sub	Prop	Obj
&V1	*type*	VENDOR
&V1	*country*	US
&V1	*homepage*	www.vendor...
&Offer1	*vendor*	&V1
&Offer1	*price*	108
	….	

Group By (*Sub*)

tg_1 = { (&V1, type, VENDOR), (&V1, *country*, US), (&V1, *homepage*, www.vendor...) }

tg_2 = { (&Offer1, *vendor*, &V1), (&Offer1, *price*, 108), (&Offer1, *product*, &P1), }

}

"Groups of Triples" or ***TripleGroups***

NTGA – Data Model

- Data model based on *nested* TripleGroups
  - ✓ More naturally capture graphs
    - TripleGroup – groups of triples sharing *Subject / Object* component
    - Can be nested at the *Object* component

```
{(&Offer1, price,  108),
 (&Offer1, vendor,  &V1),
 (&Offer1, product, &P1),
 (&Offer1, delDays,  2)
}
```

```
{(&Offer1, price,   108),
 (&Offer1, vendor, {(&V1, label,    vendor1),
                    (&V1, country, US),
                    (&V1, homepage, www.vendors….)}
 (&Offer1, product, &P1),
 (&Offer1, delDays,  2)
}
```

Computer Science
NC STATE UNIVERSITY

NTGA Operators…(1)

- `TG_Unnest` – *unnest* a nested TripleGroup

{(&Offer1, *price*, *108*),
(&Offer1, *vendor*,{(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, *www.ven..*)}
(&Offer1, *product*, &P1),
(&Offer1, *delDays*, 2)}

TG_Unnest →

{(&Offer1, *price*, *108*),
(&Offer1, *vendor*, &V1),
(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, *www.ven..*)}
(&Offer1, *product*, &P1),
(&Offer1, *delDays*, 2)}

- `TG_Flatten` – generate equivalent n-tuple

(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, www.ven...)}

TG_Flatten ↓ $\cong$ *"Content Equivalence"*

(&V1, *label*, vendor1, &V1, *country*, US, &V1, *homepage*, www.ven...)

t_1 t_2 t_3

Computer Science
NC STATE UNIVERSITY

NTGA Operators…(2)

- `TG_Join` – join between different structure TripleGroups based on join triple patterns

TG{price, vendor, delDays, product}

{ (&Offer1, *price*, 108),
(&Offer1, *vendor*, &V1),
(&Offer1, *product*, &P1),
(&Offer1, *delDays*, 2) }

TG{label, country, homepage}

(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, ww.ven...)}

?o *vendor* ?v

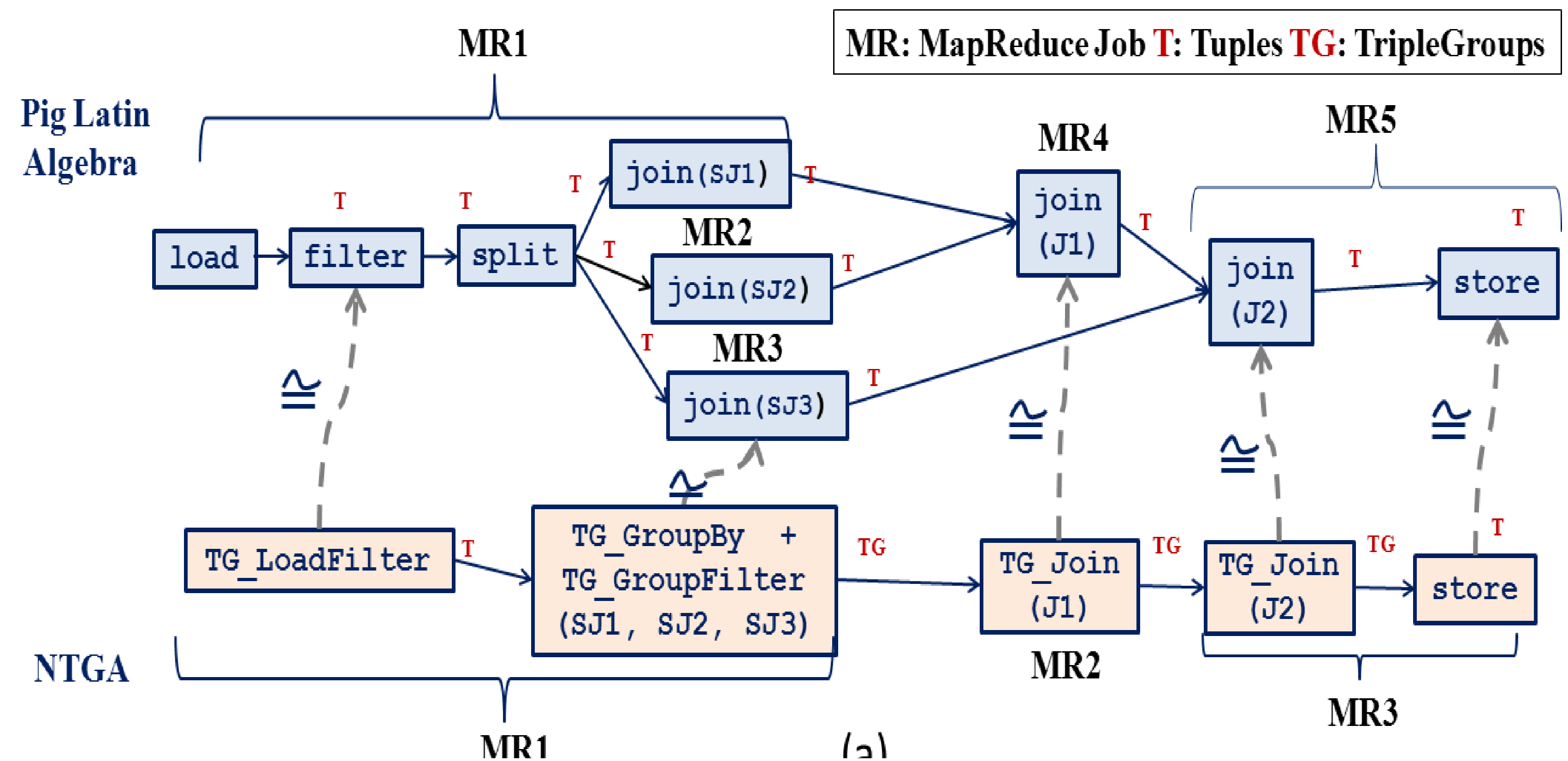
?v *country* ?vcountry

TG_Join

{(&Offer1, *price*, *108),*
(&Offer1, *vendor*, {(&V1, *label*, vendor1),
(&V1, *country*, US),
(&V1, *homepage*, *www.ven..*)}
(&Offer1, *product*, &P1),
(&Offer1, *delDays*, 2)}

RDF Data Processing using NTGA

- TripleGroups resulting from NTGA operators can be mapped to Pig's n-tupled results



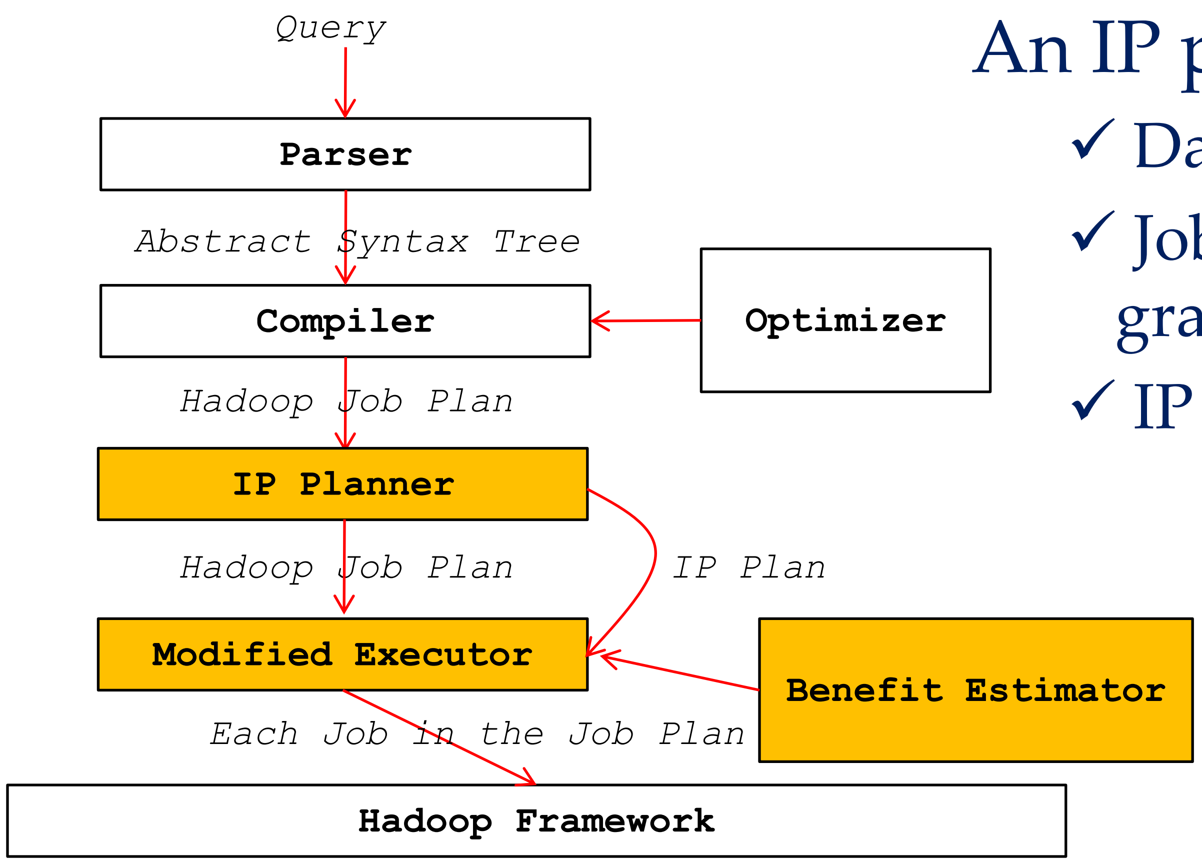
#MR cycles reduced from 5 to 3

Goal2: Minimizing Intermediate Data

- Filter out irrelevant records that may not join in subsequent phases
  - ✓ Use ***side-way information passing*** to reduce the #intermediate tuples that are loaded, sorted, and transferred in intermediate steps
- Challenge: Adapting SIP to MapReduce
  - ✓ Pipelined or Operation parallelism absent. Only partitioned parallelism support
    - Each job is blocked until the completion of a previous job
    - Which operators should generate / receive summary
    - All operators cannot run at the same time
  - ✓ Limited direct communication method between units
    - Shared memory/Message passing/TCP communication

Enabling Information-Passing in Hadoop Plans

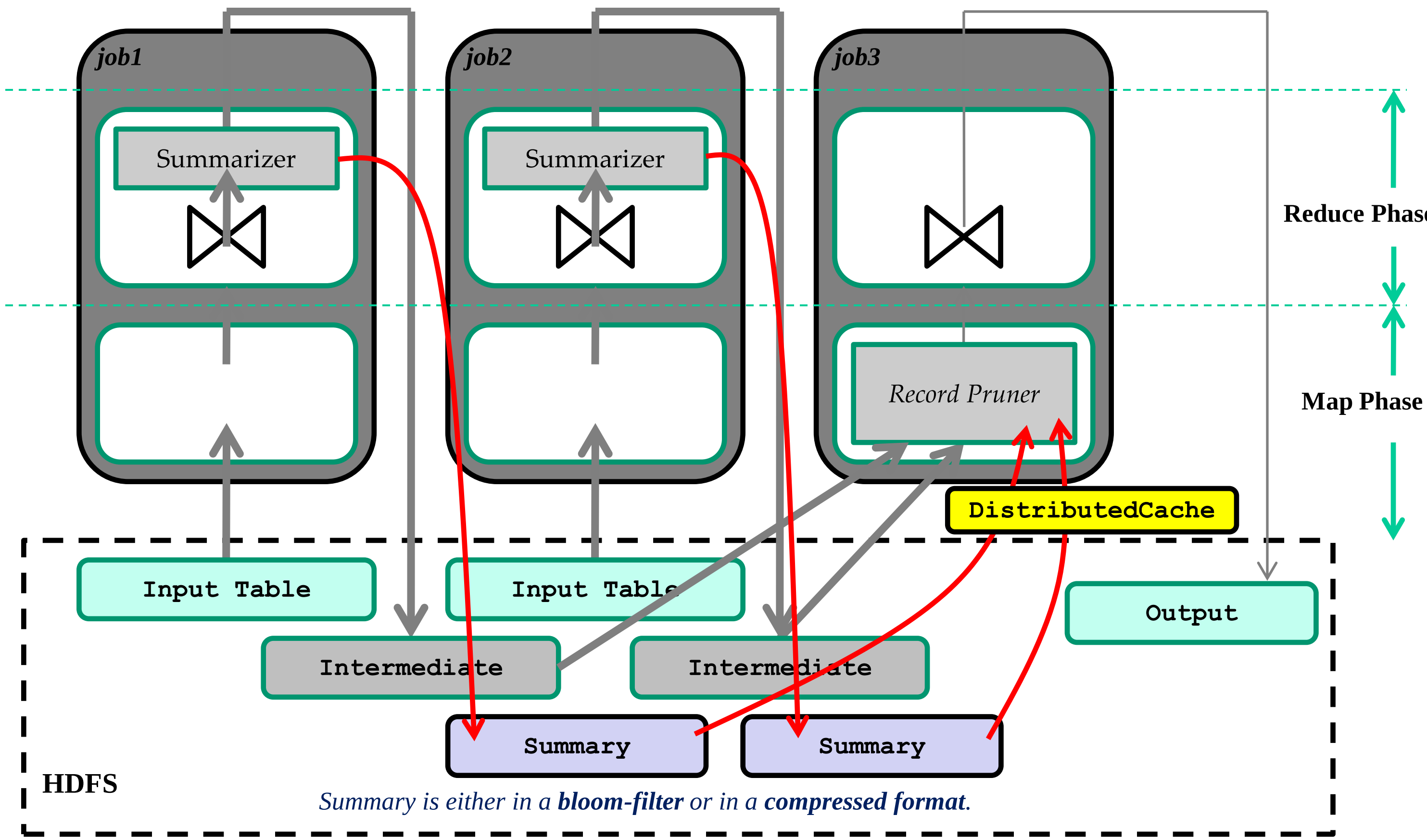
- Compile-time IP preparation



An IP plan consists of

- ✓ Dataflow graph
- ✓ Job dependency graph
- ✓ IP descriptors

Inter-Job Information Passing



Computer Science
NC STATE UNIVERSITY

Evaluation

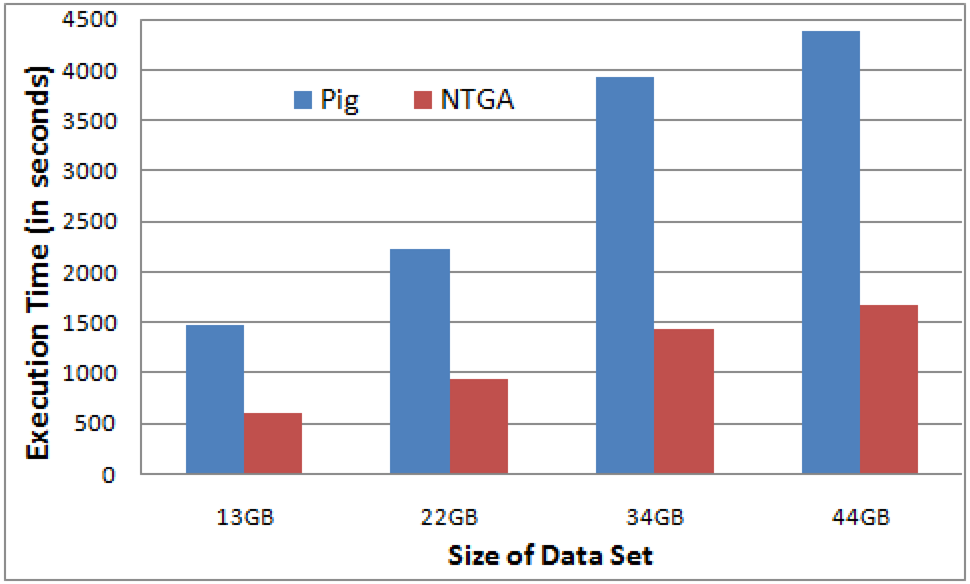
- Setup: 5-node Hadoop clusters on NCSU's Virtual Computing Lab*
- Dataset: Synthetic benchmark dataset generated using BSBM** tool
- Evaluating TripleGroup based Query Plans using ***N-triple format*** (max. 44GB – approx. 170 million triples)
  - ✓ Task A – Scalability with increasing size of RDF graphs
  - ✓ Task B –Scalability with increasing cluster sizes
  - ✓ Task C – Comparative Study of hybrid plans
- Evaluating Inter-Job Information Passing using ***SQL-dump*** (max. 50GB – 500000 products)
  - ✓ Task D – Scalability with increasing size of RDF graphs

*https://vcl.ncsu.edu

**http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/

Experimental Results…(Task A)

Cost Analysis across Increasing size of RDF graphs (5-node)



Query Pattern:
Two star-join structures
Total of 6 triple patterns
Naïve – 5 join ops
N-way join – 3 join ops
NTGA – 2 join ops

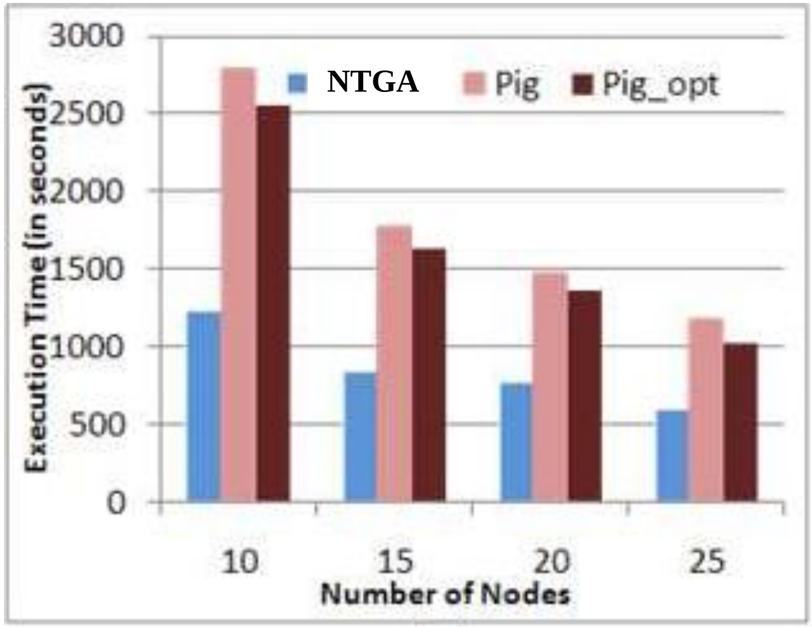
Key Observations:

✓Benefit of TripleGroup based processing seen across data sizes – up to 60% in most cases

✓ TODO - delineate different types of TripleGroups after star-joins

Experimental Results…(Task B)

Cost Analysis across Increasing Cluster Sizes



Query pattern with three star-joins and two chain-joins (32GB)

Key Observations:

✓ NTGA has 56% gain for 10-node cluster over Pig approaches

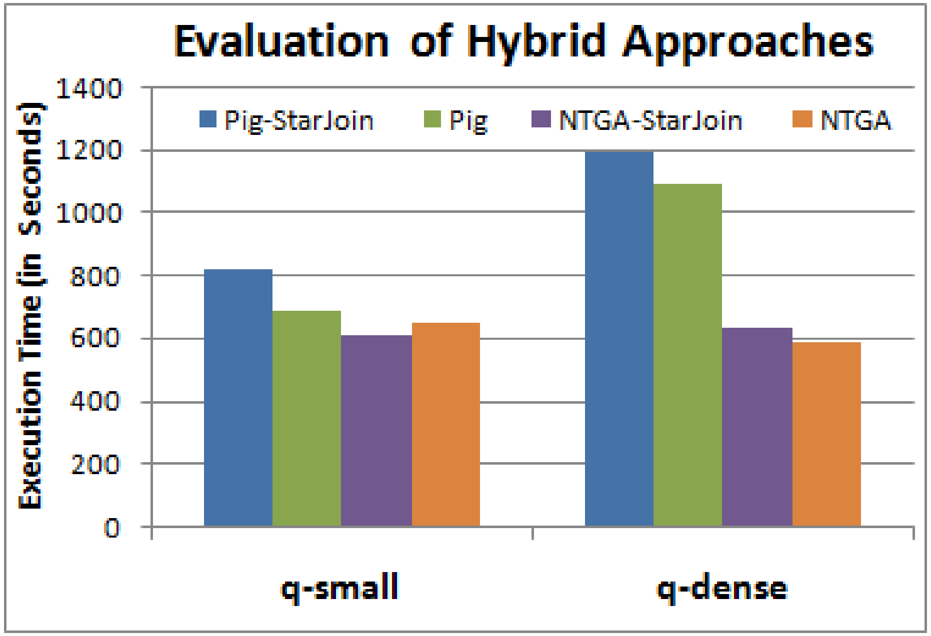
✓ Pig approaches catch up with increasing cluster size

- Increasing nodes decrease probability of disk spills with the `SPLIT` approach

✓ NTGA still maintains 45% gain across the experiments

Experimental Results…(Task C)

Comparative Study of Hybrid Plans (5-node)



Pig-StarJoin: Compute only star-joins using Pig 's `JOIN`;

NTGA-StarJoin: Compute only star-joins using NTGA's `TG_GroupBy`

Query Patterns: 3 star-joins

q-small – 1,3,1 triple patterns in each star

q-dense – 3 triple patterns in each star

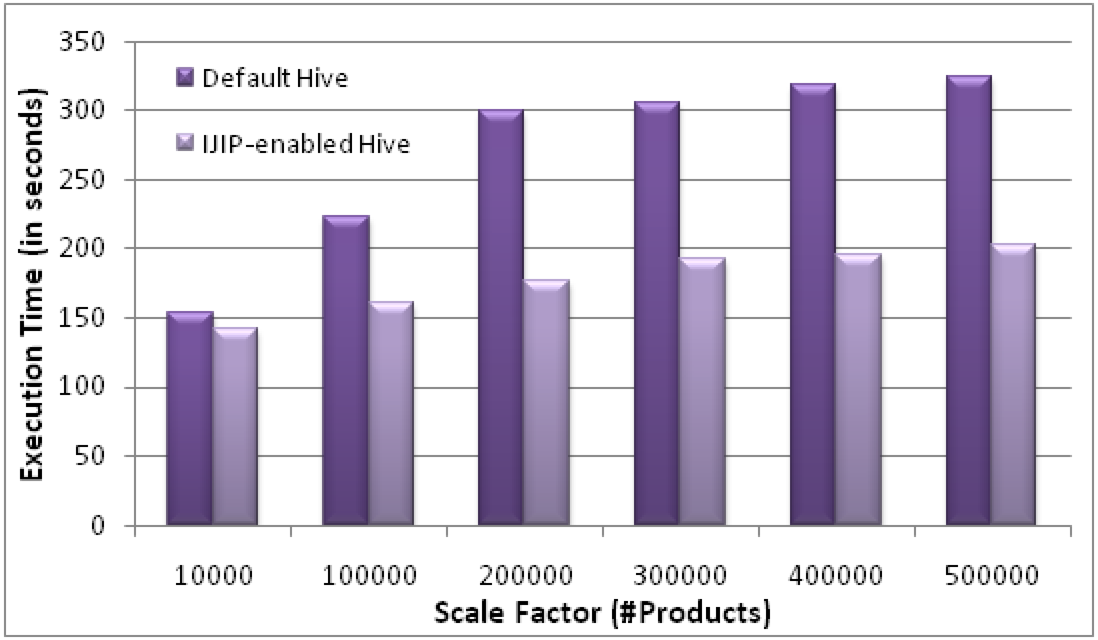
Key Observations:

- ✓ NTGA-StarJoin and NTGA have 42% to 46% performance gain over Pig
- ✓ NTGA better than NTGA-StarJoin for denser query patterns
- ✓ PigStarJoin worse than Pig due to overhead of flattening n-tuples into TripleGroups

Computer Science NC STATE UNIVERSITY

Experimental Results…(Task D)

Cost Analysis across Increasing size of RDF graphs (5-node)



Query Pattern:

-retrieves products which have two certain properties and are classified to a certain type (three joins).

-Generates summary on the output of the 2nd join and 3rd job prunes records by using the summary.

Key Observations:

✓IP-enabled Hive shows more than 35% performance improvement in terms of execution time

Related Work

MapReduce-based Processing

- High-level Dataflow Languages:
  - Pig Latin[Olston08], [HiveQL], [JAQL]
- Graph Pattern Matching
  - HadoopRDF[Husain10], SHARD[Rohloff10], [Huang11], RDF-Molecules([Newman08], [Hunter08])
- Efficient Join Techniques
  - Map-Reduce-Merge[Yang07], [Afrati10], Hadoop++ [Dittrich10], Log Processing[Blanas10]
- DB/MR Hybrid Architecture
  - HadoopDB [Abadi09]
- Reasoning
  - [Urbani07]

Conclusion

- Generalized query plan strategy for efficient processing of RDF data
  - ✓ *TripleGroup based processing* to minimize #MR cycles
  - ✓ *Inter-job information passing* to minimize intermediate data
- → Future work:
  - → Support for *inferencing* e.g. sameAs for multiple support datasets and subsumption hierarchies
  - → Compression of URIs
  - → Integrating both strategies in the same system

Computer Science
NC STATE UNIVERSITY

References

[Dean04] Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51 (2008) 107–113

[Olston08] Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proc. International Conference on Management of data. (2008)

[Abadi09] Abouzied, A., Bajda-Pawlikowski, K., Huang, J., Abadi, D.J., Silberschatz, A.: Hadoopdb in action: building real world applications. In: Proc. International Conference on Management of data. (2010)

[Newman08] Newman, A., Li, Y.F., Hunter, J.: Scalable semantics: The silver lining of cloud computing. In: eScience. IEEE International Conference on. (2008)

[Hunter08] Newman, A., Hunter, J., Li, Y., Bouton, C., Davis, M.: A scale-out RDF molecule store for distributed processing of biomedical data. In: Semantic Web for Health Care and Life Sciences Workshop. (2008)

[Urbani07] Urbani, J., Kotoulas, S., Oren, E., Harmelen, F.: Scalable distributed reasoning using mapreduce. In: Proc. International Semantic Web Conference. (2009)

[Abadi07] Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007

[Dittrich10] Dittrich, J., Quiane-Ruiz, J., Jindal, A., Kargin, Y., Setty, V., Schad, J.: Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). VLDB 2010/PVLDB

[Yang07] Yang, H., Dasdan, A., Hsiao, R., Parker Jr., D.S.: Map-Reduce-Merge: simplified relational data processing on large clusters. SIGMOD 2007

[Afrati10] Afrati, F.N., Ullman, J.D.: Optimizing joins in a map-reduce environment. In: Proc. International Conference on Extending Database Technology. (2010)

[Husain10] Husain, M., Khan, L., Kantarcioglu, M., Thuraisingham, B.: Data intensive query processing for large RDF graphs using cloud computing tools. In: Cloud Computing (CLOUD), IEEE International Conference on. (2010)

[Huang11] Jiewen Huang, Daniel J. Abadi, and Kun Ren. Scalable SPARQL Querying of Large RDF Graphs. Proceedings of the VLDB Endowment, 4(11), 2011.

[Rohloff10] Kurt Rohloff and Richard E. Schantz. High-performance, Massively Scalable Distributed Systems using the MapReduce Software Framework: the SHARD Triple-store. In Programming Support Innovations for Emerging Distributed Applications, pages 4:1–4:5, 2010.

[Blanas10] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, and Yuanyuan Tian. A Comparison of Join Algorithms for Log Processing in MapReduce. In Proc. International conference on Management of Data, 2010

[HiveQL] http://hadoop.apache.org/hive/

[JAQL], http://code.google.com/p/jaql

Computer Science
NC STATE UNIVERSITY

Thank You!

Computer Science
NC STATE UNIVERSITY

Possible Optimizations (1)

Issues

✓ SPLIT operator for RDF data,

#sub flows = #unique property types

might be a large number of subflows

→Concurrent sub flows compete for memory resources

→Higher risk of disk spills → increased I/O costs

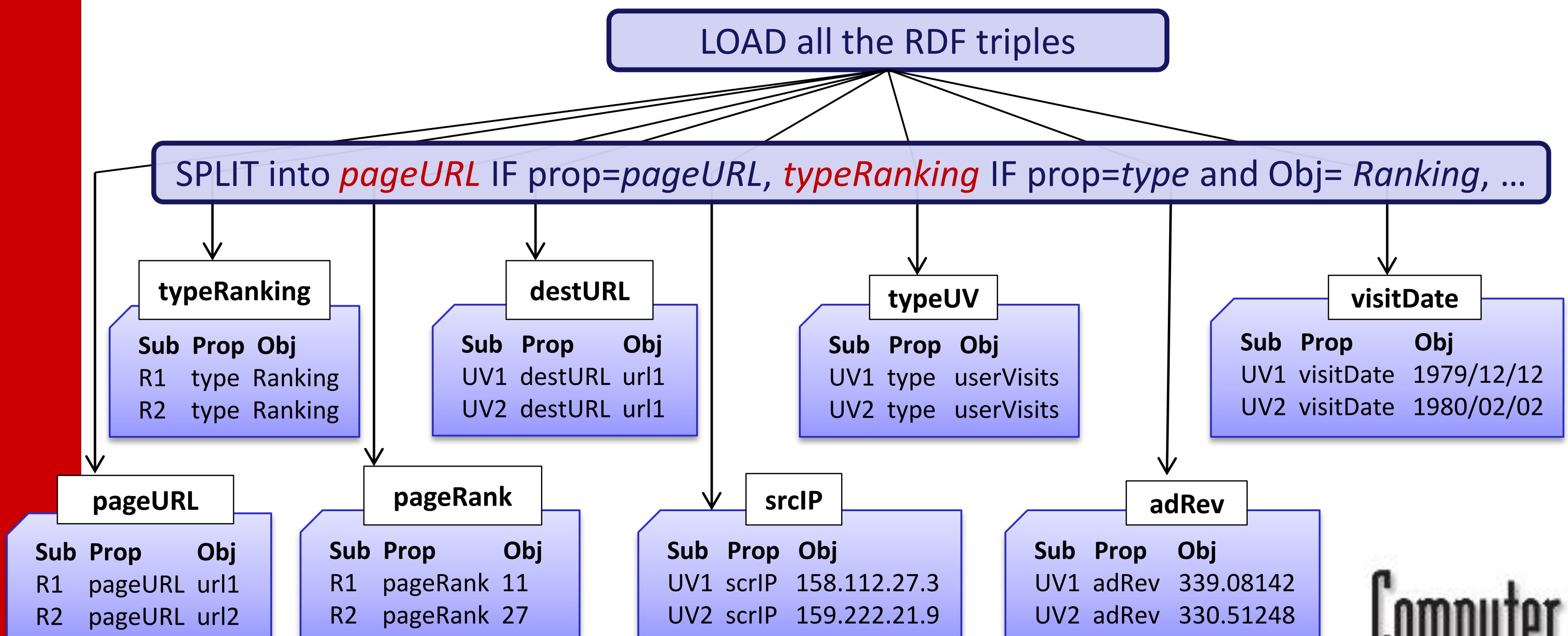
Our Approach : RAPID+

- **Goal :** Minimize I/O and communication costs by reducing MR cycles
- Reinterpret and refactor operations into a more suitable (coalesced) set of operators – NTGA algebra
- **Foundation:**
  - ✓ Re-interpret multiple star joins as a *grouping* operation
    - ✓ leads to "groups of Triples" (TripleGroups) instead of n-tuples
    - ✓ different structure BUT "content equivalent"
  - ✓ NTGA- algebra on TripleGroups

Computer Science
NC STATE UNIVERSITY

Possible Optimizations (1)

- Vertical Partitioning (VP) in 1 MR cycle
  - Input file read only once → better!!
  - ✓ In Pig Latin, VP can be achieved using the *SPLIT* operator



Computer Science
NC STATE UNIVERSITY

NTGA Operators…(2)

- `TG_GroupFilter` – retain only TripleGroups that satisfy the required query sub structure

  → *Structure-based filtering*

TG

{ (&V1, *label*, vendor1),
 (&V1, *country*, US),
 (&V1, *homepage*, www.ven..) },

{ (&Offer1, *price*, 108),
 (&Offer1, *vendor*, &V1),
 (&Offer1, *product*, &P1),
 (&Offer1, *delDays*, 2) } ,

{ (&Offer2, *vendor*, &V2),
 (&Offer2, *product*, &P3),
 (&Offer2, *delDays*, 1) } }

TG_GroupFilter (TG, {*price, vendor, delDays, product*}) →

$\mathbf{TG}_{\{price,\ vendor,\ delDays,\ product\}}$

{ (&Offer1, *price*, 108),
 (&Offer1, *vendor*, &V1),
 (&Offer1, *product*, &P1),
 (&Offer1, *delDays*, 2) }

Eliminate TripleGroups with missing triples (edges)

Computer Science
NC STATE UNIVERSITY

NTGA Operators…(3)

- `TG_Filter` – filter out triples that do not satisfy the filter condition (FILTER clause)

  → *Value-based filtering*

TG$_{\{price,\ vendor,\ delDays,\ product\}}$

{ (&Offer1, *price*, 108),
(&Offer1, *vendor*, &V1),
(&Offer1, *product*, &P1),
(&Offer1, *delDays*, 2) } ,

{ (&Offer3, *vendor*, &V2),
(&Offer3, *product*, &P3),
(&Offer3, *price*, 306),
(&Offer3, *delDays*, 1) } }

TG_Filter$_{price<200}$ **(TG)** →

TG$_{\{price,\ vendor,\ delDays,\ product\}}$

{ (&Offer1, *price*, 108),
(&Offer1, *vendor*, &V1),
(&Offer1, *product*, &P1),
(&Offer1, *delDays*, 2) }

Eliminate TripleGroups with triples that do not satisfy filter condition

UPDATE

- Additional evaluation –
  - ✓ Up to 65% performance gain on synthetic benchmark dataset* for three/two star-join queries
  - ✓ Experiment extended to 30-node clusters with 1 billion 3-ary triples (43GB) – 41% gain
- RAPID+ now includes a SPARQL interface
- Join us for a demo of RAPID+@VLDB2011*

*Kim, H., Ravindra, P., Anyanwu, K : *From SPARQL to MapReduce: The Journey using a Nested TripleGroup Algebra*. To appear In: Proc. International Conference on Very Large Data Bases. (VLDB 2011)

Computer Science
NC STATE UNIVERSITY

Environment

- Node Specifications
  - Single / duo core Intel X86
  - 2.33 GHz processor speed
  - 4G memory
  - Red Hat Linux
- Pig 0.8.0
- Hadoop 0.20
  - Block size 256MB

Experiment Results

Percentage Performance Gain

$$= \frac{(\text{exec time 1}) - (\text{exec time 2})}{(\text{exec time 1})}$$

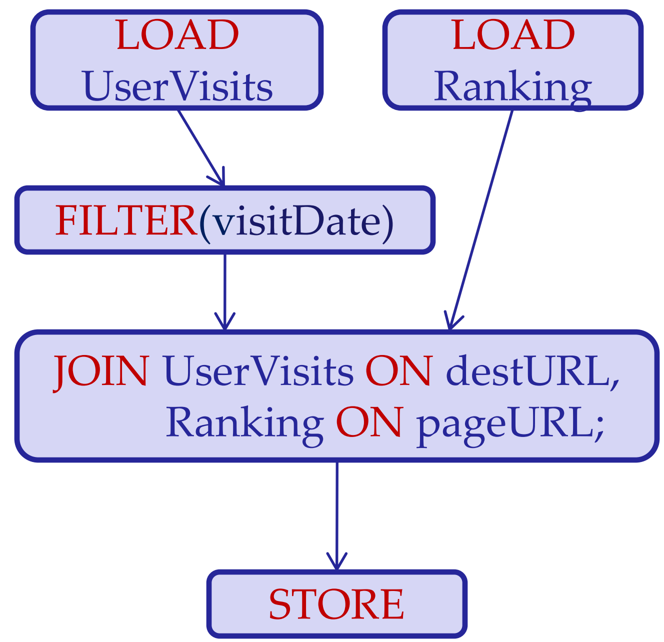
Structured Data Processing in Pig

UserVisits

srcIP	destURL	visitDate	adRevenue	…
158.112.27.3	url1	1979/12/12	339.08142	….
158.112.27.3	url5	1979/12/15	180.334	….
150.121.18.6	url1	1979/12/28	550.7889	….
…	…	…	…	…

Ranking

pageRank	pageURL	avgDur
11	url1	96
23	url2	3
18	url3	87
…	…	…



Query: *Retrieve the pageRank and adRevenue of pages visited by particular users between "1979/12/01" and "1979/12/30"*

JOIN: Pig Latin → MapReduce

UserVisits

	srcIP	destURL	visitDate	adRev	…
url1	158.112.27.3	url1	1979/12/12	339.081	…
url2	158.112.27.3	url2	1979/12/15	180.334	…
url1	150.121.18.6	url1	1979/12/28	550.78	…

Ranking

	pageRank	pageURL	avgDur
url1	11	url1	96
url2	23	url2	3

map

Annotate based on join key

JOIN UserVisits ON destURL, Ranking ON pageURL;

reduce

Package tuples

Reducer 1

url1				
158.112.27.3	url1	url1	11	…
150.121.18.6	url1	url1	11	…

Reducer 2

url2				
158.112.27.3	url2	url2	3	…

…	srcIP	destURL	pageURL	pageRank	…
…	158.112.27.3	url1	url1	339.081	…
…	150.121.18.6	url1	url1	550.78	…
…	158.112.27.3	url2	url2	180.334	…

Background

- Hadoop Join Processing Techniques
  - Standard Repartitioning Join
  - Fragment-Replication Join
  - Map-Merge Join

Background

- Hadoop Join Processing Techniques (Cont.)
  - Fragment-Replication Join and Map-Merge Join
    - Alternative Join techniques
    - Process join operation in map-phase
      - Can remove the cost to sort and transfer data between phases
    - Used in very restricted ways
      - In the presence of pre-processing or
      - One of the two input relation is small enough to be buffered in available memory)