Data Intensive Query Processing for Large RDF Graphs Using Cloud Computing Tools

Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing

Doruk Özokan

About...

We need to answer queries on large RDF graphs efficiently.

Efficiency:

- Run as quick as possible.
- Answer concurrent requests without performance penalty.
- Handle continuously growing datasets.

Today, there are solutions for querying RDF datasets. Ex: Jena, Sesame...

But, limited capabilities...

Why a new framework?

- Current frameworks (Jena, Sesame...) do not scale for large RDF graphs.

  They are designed for a single machine scenario. And they are weak for terrabytes of data.

  Ex: 10 million triples can be processed in Jena in-memory model running on a machine uses 2GB of memory.

- Heavy I/O operations cannot be handled on a single machine...

New Technologies - Cloud Computing

Cloud technology is very promising for handling scalability requirements.

Hadoop is suitable for keeping large amounts of data.

Hadoop

- Distributed file system
- Files are saved with replication, consistency guaranteed.
- It's fault tolerant.

Hadoop uses MapReduce programming model.

MapReduce

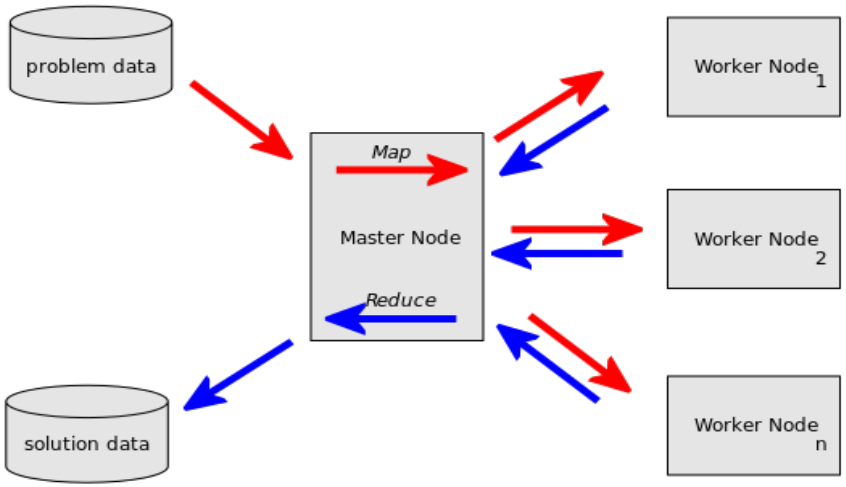
A programming model for large data sets.

Distributed computing on clusters of computers.

Map: Take the input, divide it into smaller sub-problems, and distribute them to worker nodes.

Reduce: Collect the answers to all the sub-problems and combine them in some way to form the output.

They introduce a new framework based on this technology to work on large datasets.



How Hadoop/MapReduce is used?

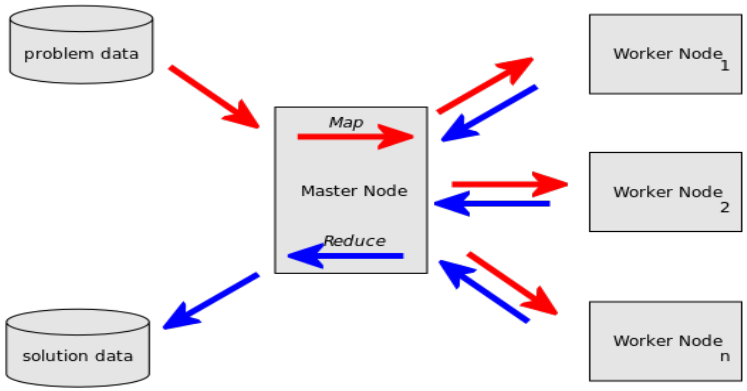
- Map for Selection
- Reduce for Join

Map Input phase (MI): Reads triple patterns from input files based on selection.

Map Output phase (MO): The selection of data is done and the output is written into local disks.

Reduce Input phase (RI): The triples are read from local disks via HTTP.

Reduce Output Phase (RO): The RO phase deals with performing the joins.



The Idea

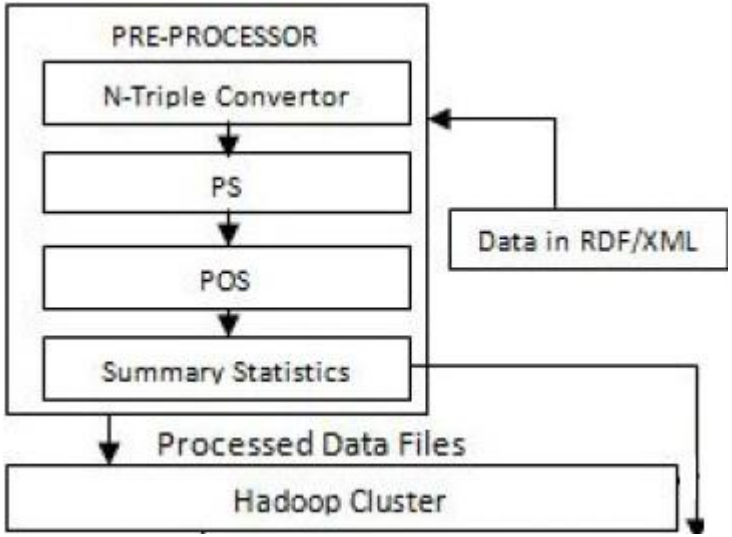
1. Convert RDF/XML into plain triple files and store using Hadoop.
2. Run triple selection tasks on nodes.
3. Fetch resultant data from nodes to the Master Node.
4. Join the results.
5. Return the result to the user.

System Architecture

Two main components:

1- Data Preprocessing Component (Store & Prepare Data)

2- Query answering component (Answer query)



Data Preprocessing Component

N-Triples Converter:

Converts RDF/XML to N-Triples.

PS Component:

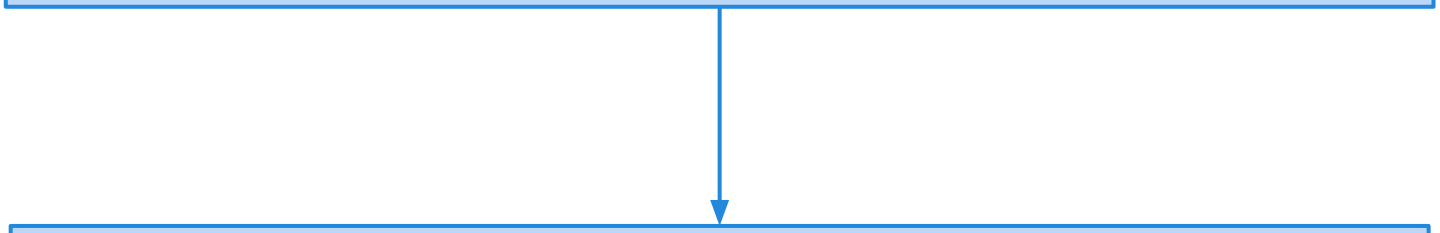
Takes N-Triples and splits them into predicate files.

POS Component:

Splits predicate files into smaller files based on type of the objects.

N-Triples Converter

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.w3.org/2001/sw/RDFCore/ntriples/">
    <dc:creator>Art Barstow</dc:creator>
    <dc:creator>Dave Beckett</dc:creator>
    <dc:publisher rdf:resource="http://www.w3.org/"/>
  </rdf:Description>
</rdf:RDF>
```

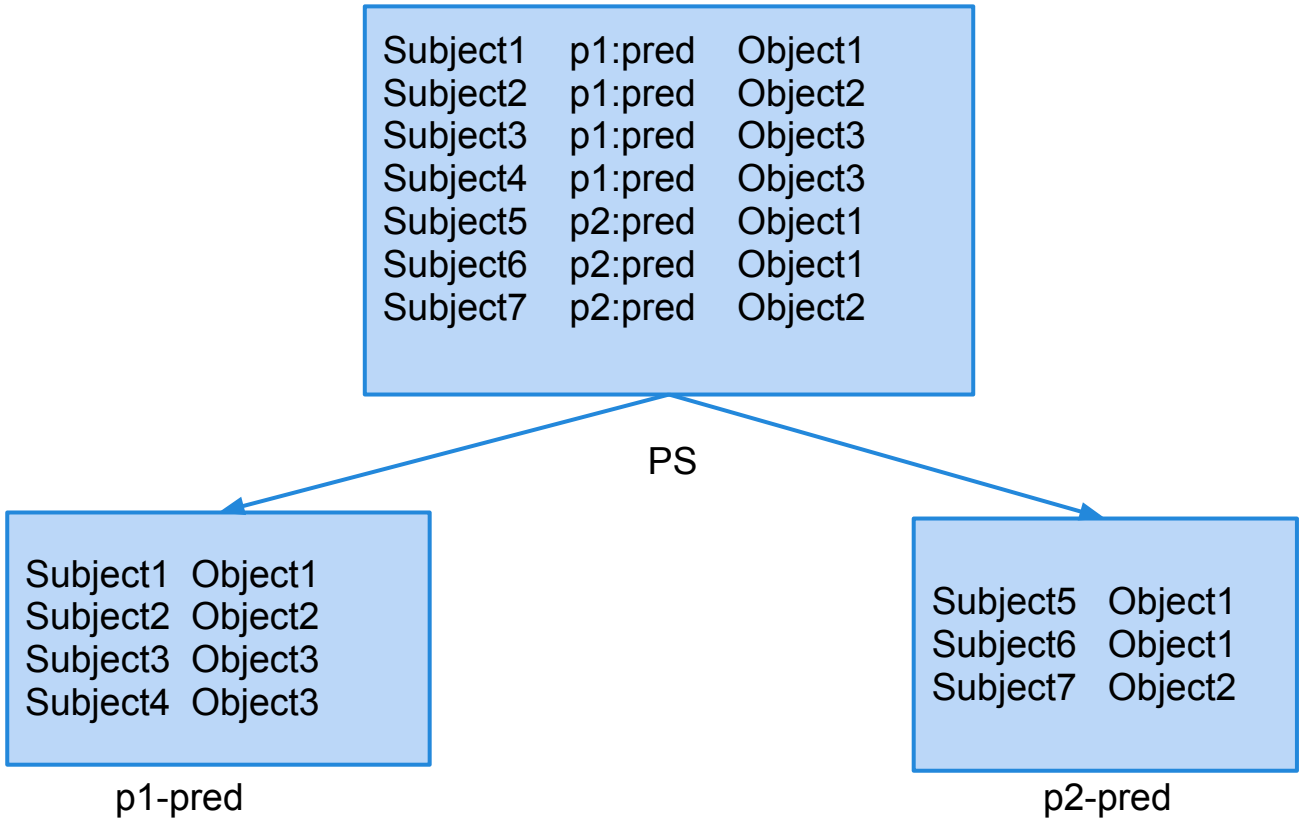


```
<http://www.w3.org/2001/sw/RDFCore/ntriples/>   <http://purl.org/dc/elements/1.1/creator>     "Dave Beckett" .
<http://www.w3.org/2001/sw/RDFCore/ntriples/>   <http://purl.org/dc/elements/1.1/creator>     "Art Barstow" .
<http://www.w3.org/2001/sw/RDFCore/ntriples/>   <http://purl.org/dc/elements/1.1/publisher>   <http://www.w3.org/>
```

PS Component

Predicate Split:

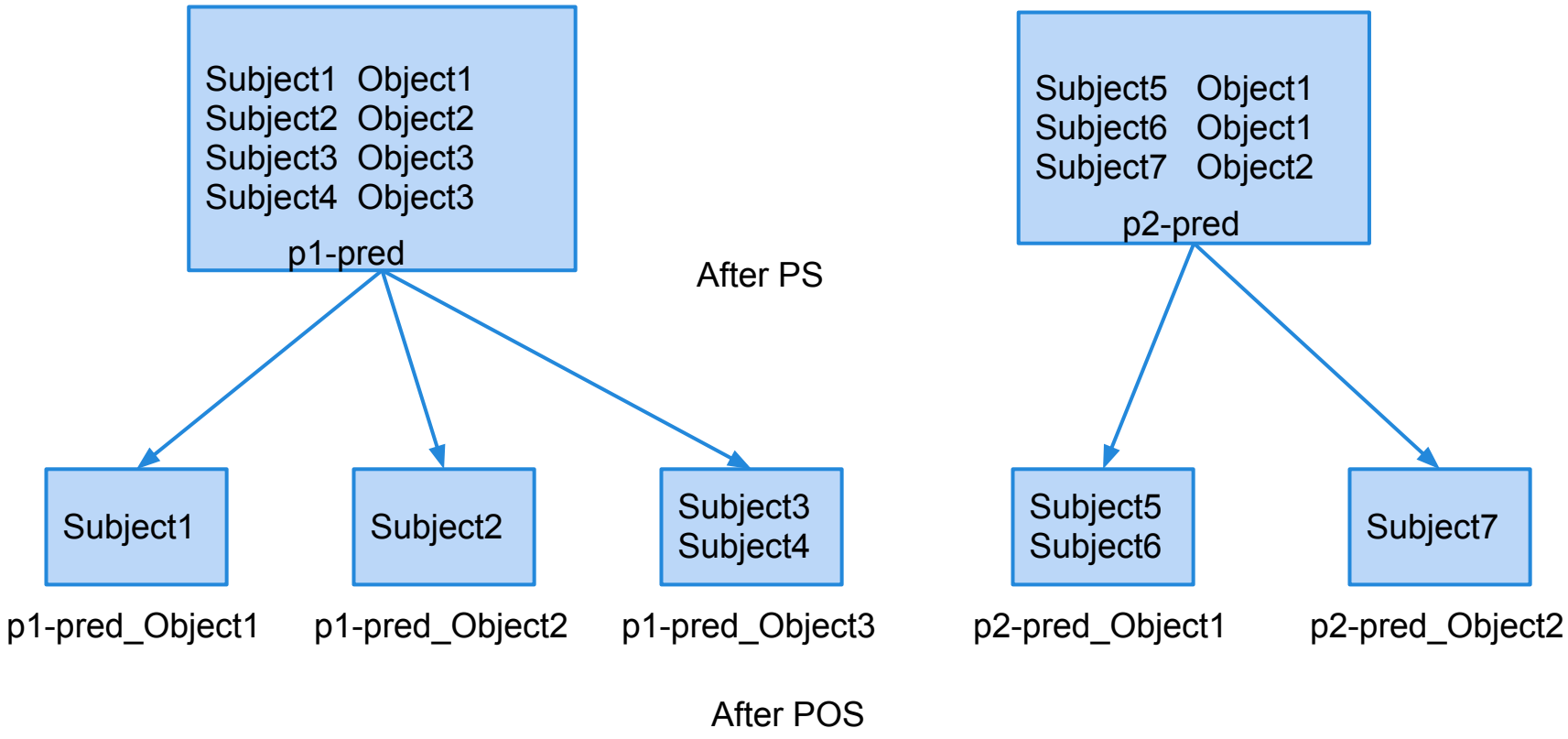
Takes N-Triples and splits them into predicate files.



POS Component

Predicate Object Split:

Split predicate files into smaller files based on type of the objects.



Data Preprocessing Component

Benefits of the storage model:

1- Small sized files. Faster to read.

2- Save space.

LUBM dataset : Benchmark datasets designed to enable researchers to evaluate their semantic web repository performances.

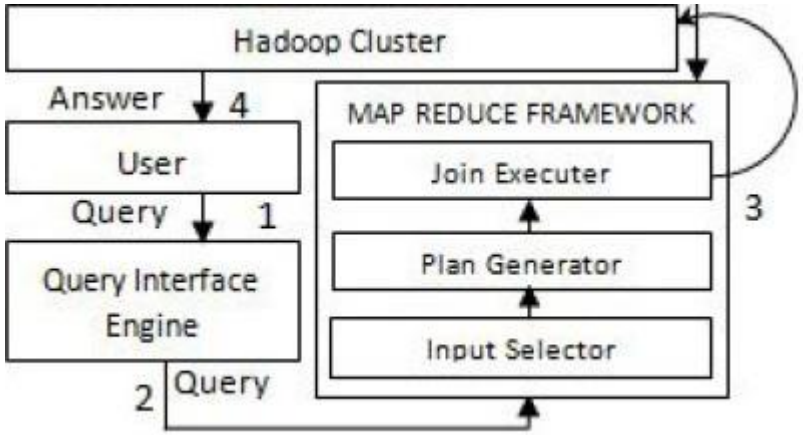
Step	Files	Size (GB)	Space Gain
N-Triples	20020	24	-
PS	17	7.1	70.42%
POS	41	6.6	7.04%

System Architecture

Two main components:

1- Data Preprocessing Component (Store & Prepare Data)

2- Query answering component (Answer query)



Query answering component

Input Selector:

Selects the necessary files for running the specified query.

Plan Generator:

Decides necessary MapReduce jobs.

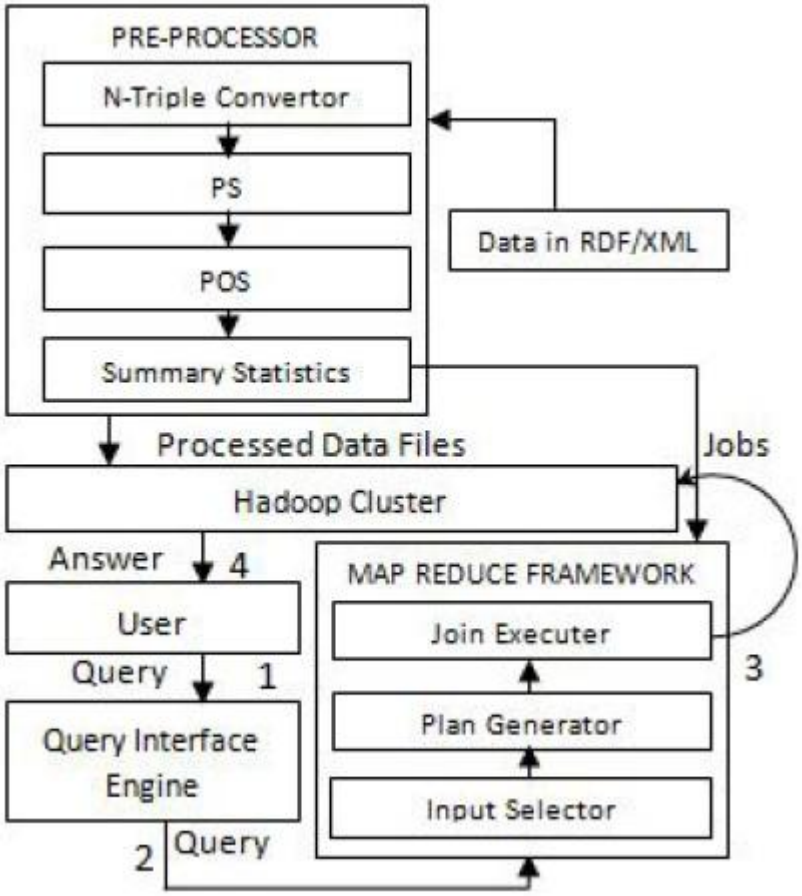
Join Executer:

Runs the jobs using MapReduce framework.

System Architecture

Prepare Data

Run Query



Definitions

MapReduce job

The unit of computation in MapReduce model.

Each job may involve one or more joins.

Triple pattern graph (TPG)

- TPG is used for representing the triple patterns which take part in joins.
- Each vertex is a triple pattern.
- Each edge represents a join between these TP on a shared variable.

Join Graph (JG)

- It is used to run coloring algorithm to decide how to run joins in a job.
- JG is constructed from TPG.
- Each vertex represents an edge of TPG.
- If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

Definitions

Conflicting MapReduce Joins:

A pair of joins on different variables sharing a triple pattern.

Nonconflicting MapReduce Joins:

A pair of joins either not sharing a triple pattern or sharing it on the same variable.

Example:

```
    SELECT ?X WHERE f
1   ?X rdf : type ub:Chair .
2   ?Y rdf : type ub:Department .
3   ?X ub : worksFor ?Y .
4   ?Y ub : subOrganizationOf <http://www.U0.edu>
```

1-3-4 is a conflicting join

For many queries, one job is not enough. In Hadoop, the jobs to perform joins cannot communicate with each other. So, joins dependent on other joins cannot be executed in the same job.

BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

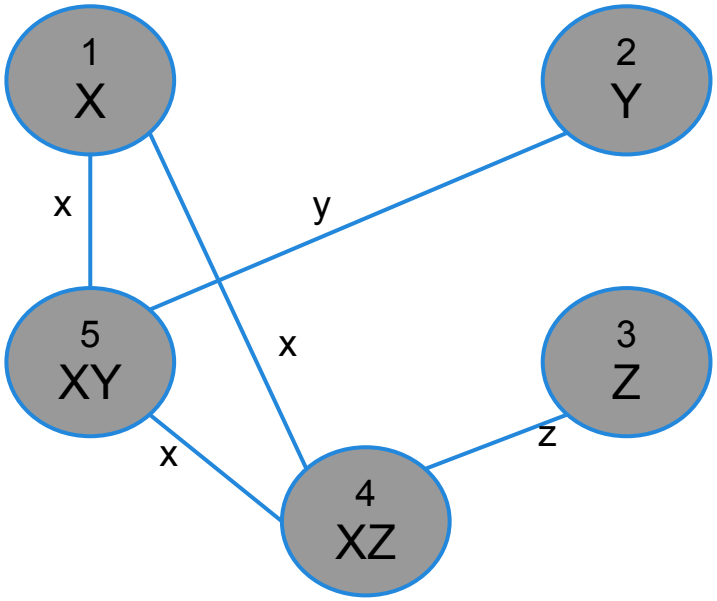
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

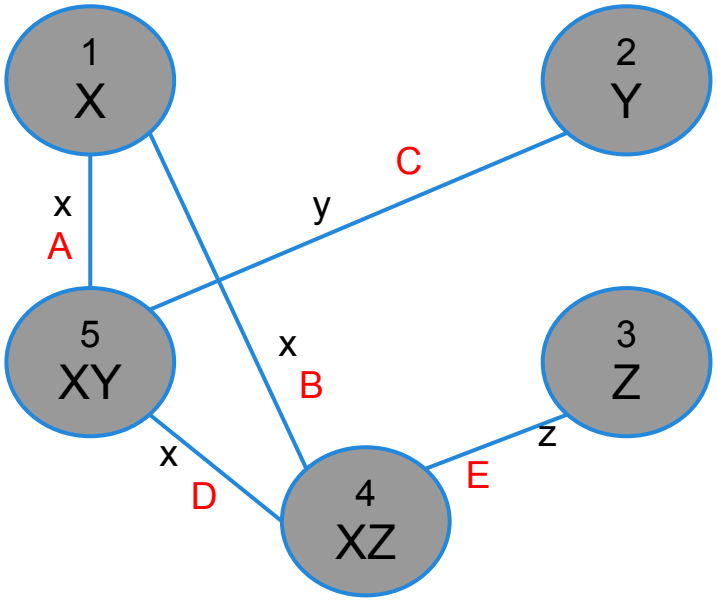
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

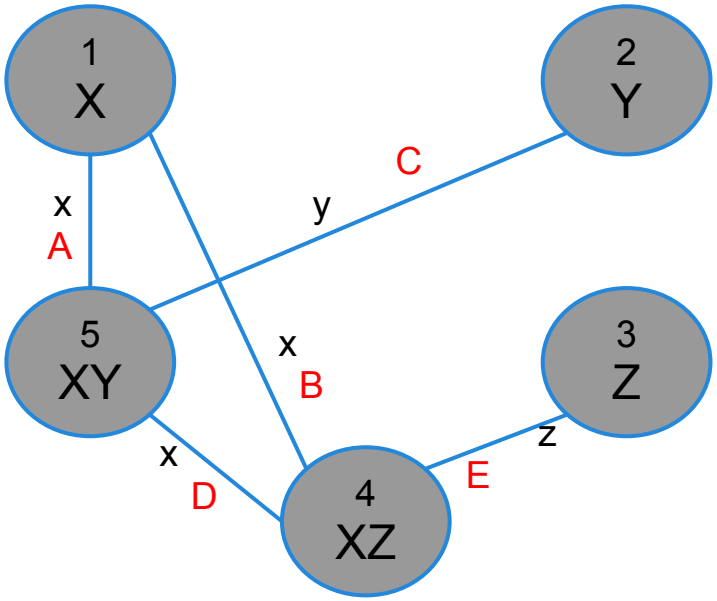
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

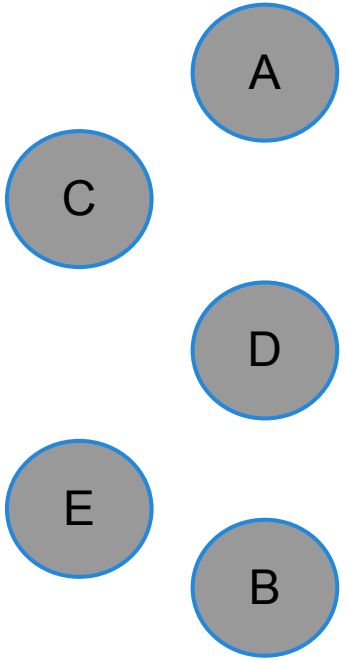
Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



JG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

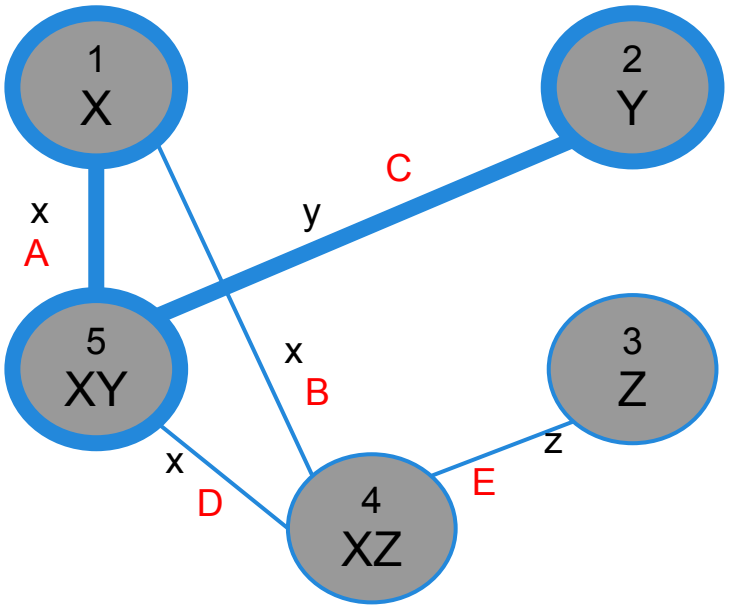
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

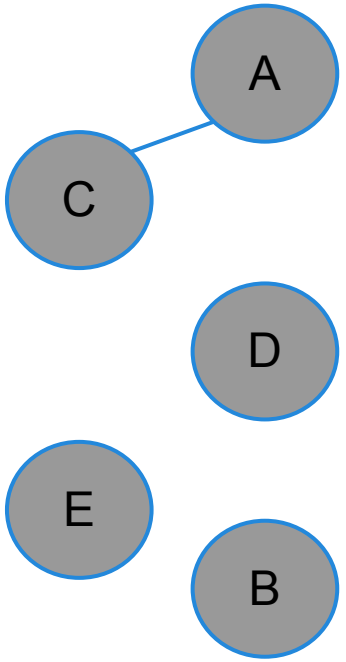
Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



JG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

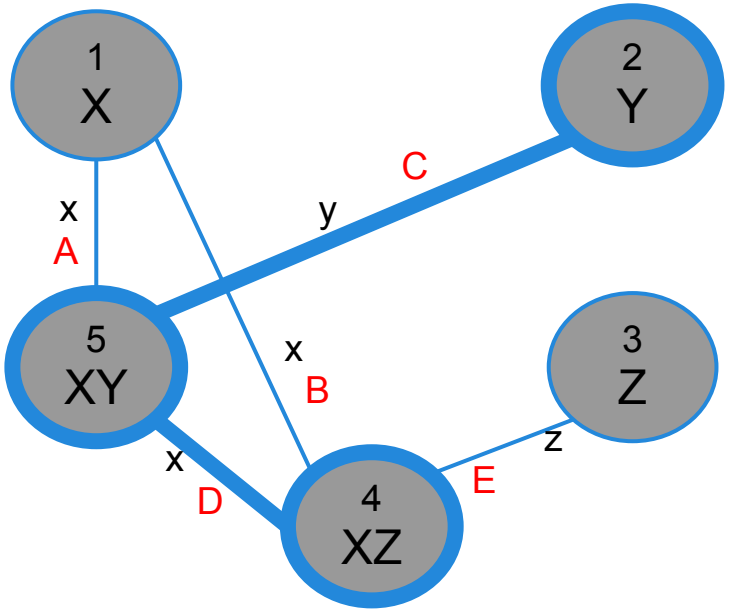
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

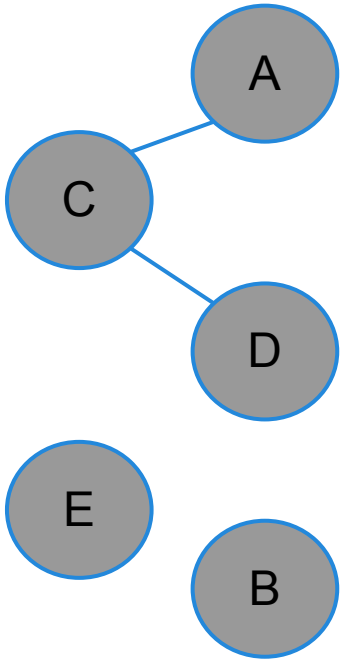
Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



JG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

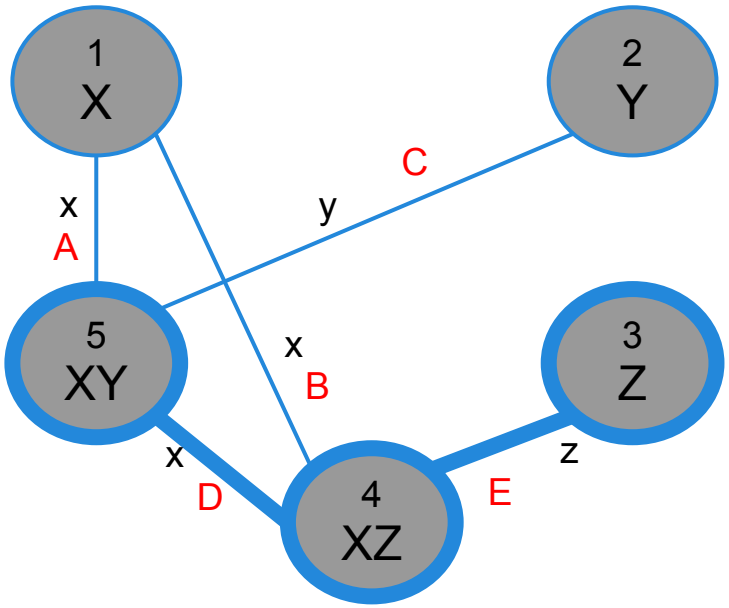
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

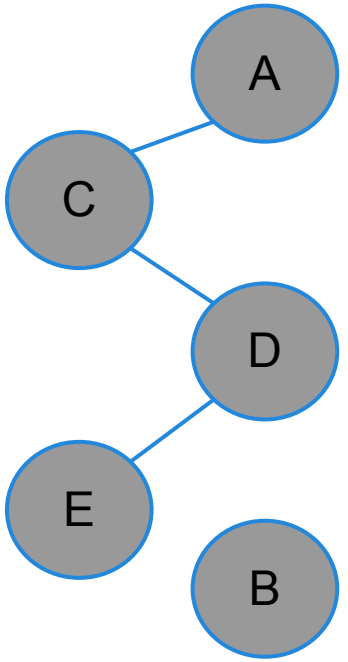
Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



JG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

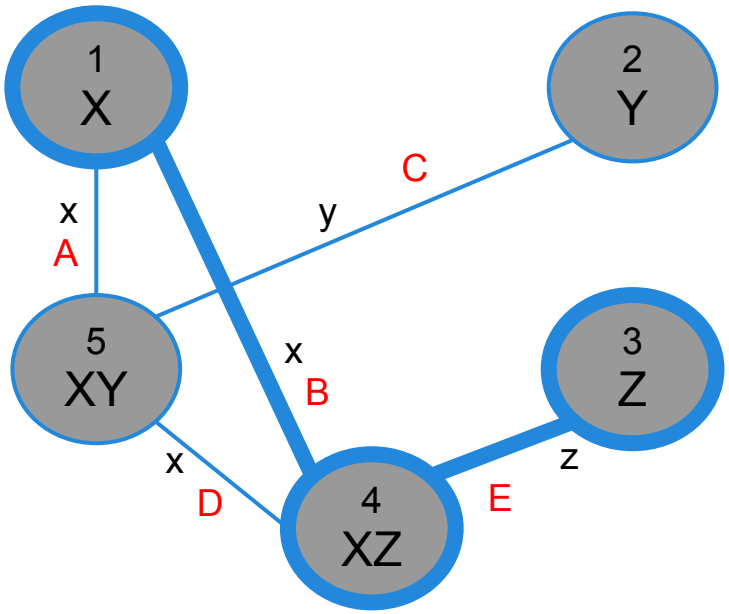
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

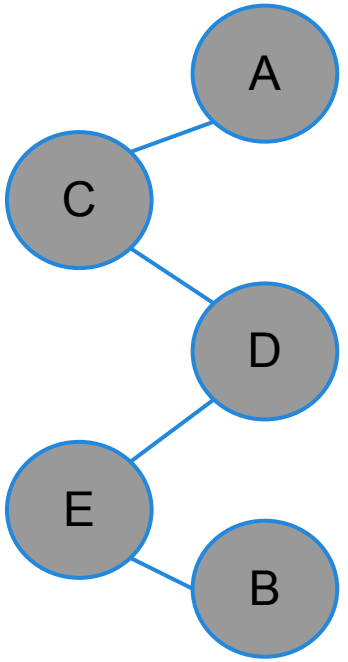
Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



JG



BestPlan - How to generate a query plan

SELECT **?V, ?X, ?Y, ?Z** WHERE{

1 **?X** rdf:type ub:GraduateStudent

2 **?Y** rdf:type ub:University

3 **?Z ?V** ub:Department

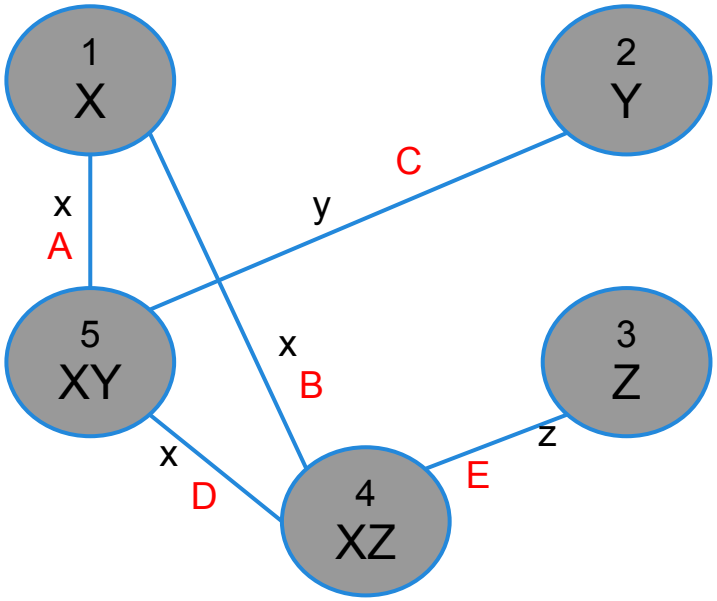
4 **?X** ub:memberOf **?Z**

5 **?X** ub:undergraduateDegreeFrom **?Y**}

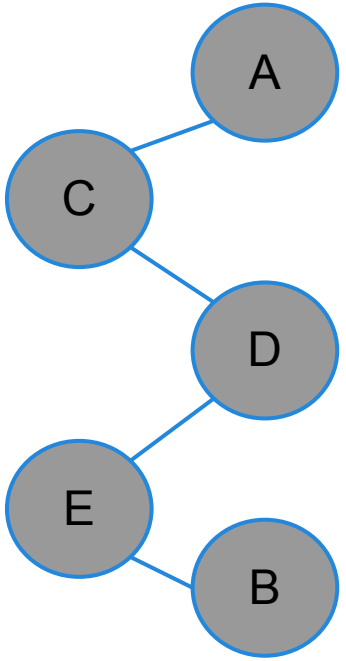
Triple Pattern Graph (TPG): Is a graph, each vertex is a TP, and each edge represents a join between these TP on a shared variable.

Join Graph (JG): Is a graph, constructed from TPG, and each vertex represents an edge of TPG.
* If two vertices on the TPG share a vertex and their variable sets are different, an edge between the nodes is drawn.

TPG



JG

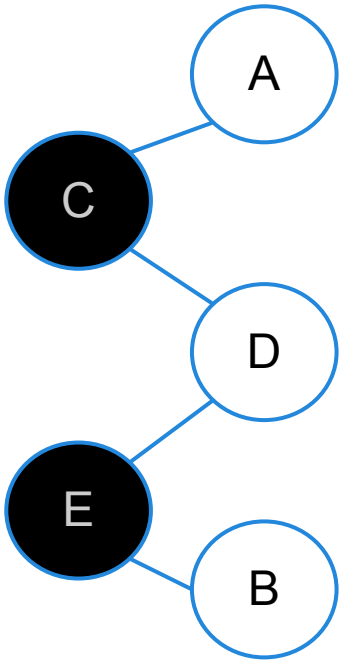


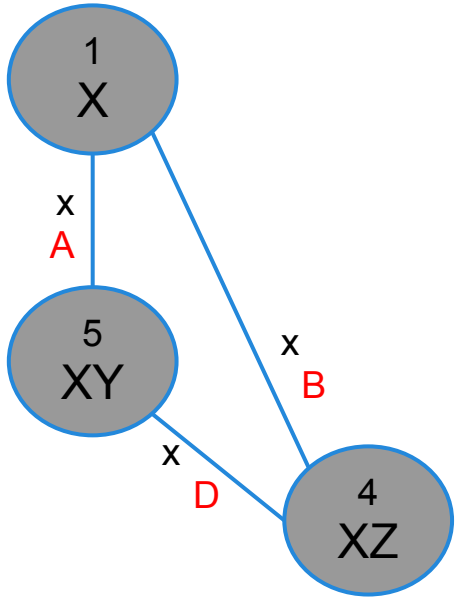
Coloring Example 1 - Step 1

If we consider a join in a job, we color it WHITE, otherwise BLACK.

Constraint: Two adjacent nodes of Join Graph cannot be WHITE at the same time.
(Remark: Joins dependent on other joins cannot be executed in the same job.)

JG



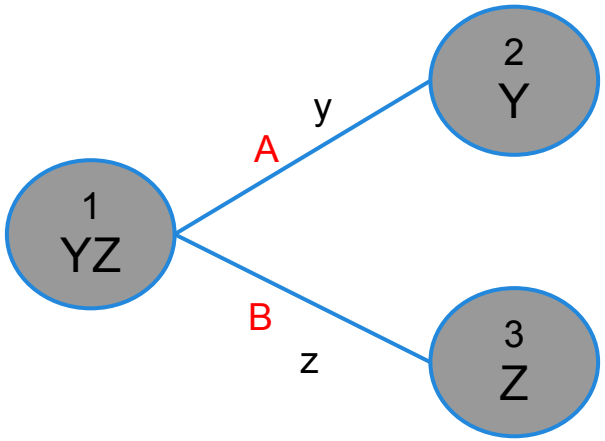


Job1: Join (X, XY, XZ)

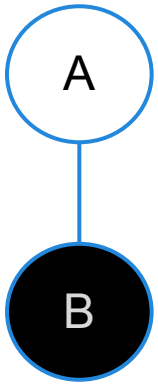
Coloring Example 1 - Step 2&3

Step 2

TPG



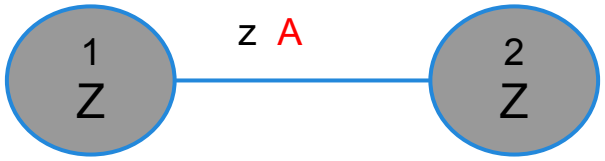
JG



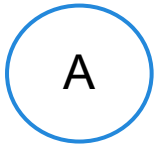
Job2: Join (Y, YZ)

Step 3

TPG



JG



Job3: Join (Z, Z)

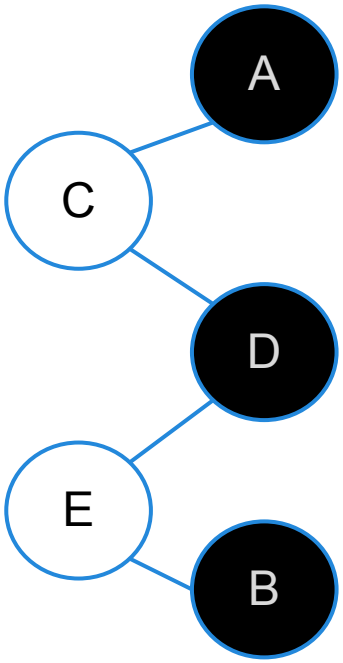
Query Plan: {Job1,Job2,Job3} = {Join(X, XY, XZ), Join(Y, YZ), Join(Z, Z)}

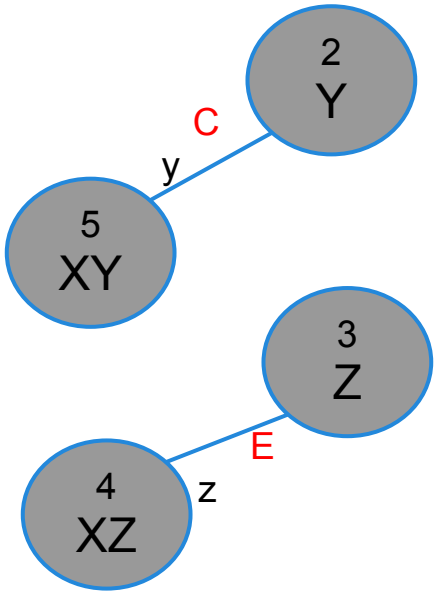
Coloring Example 2 - Step 1

If we consider a join in a job, we color it WHITE, otherwise BLACK.

<u>Constraint:</u> Two adjacent nodes of Join Graph cannot be WHITE at the same time.

JG



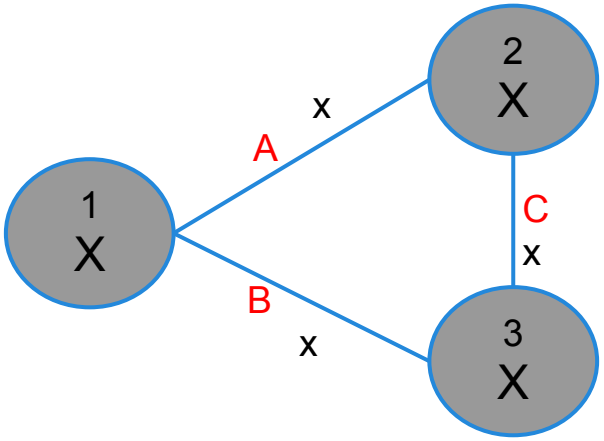


Job1: Join(XY, Y)
&
Join(XZ, Z)

Coloring Example 2 - Step 2

Step 2

TPG

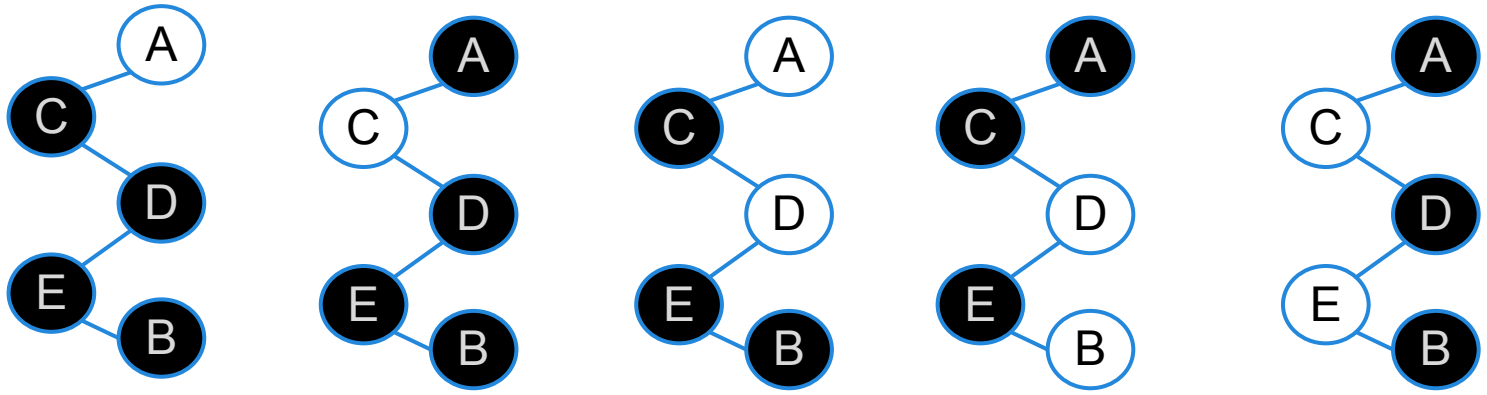


Job2: Join(X, X, X)

Query Plan: {Job1,Job2} = {[Join(XY, Y), Join(XZ, Z)], Join(X, X, X)}

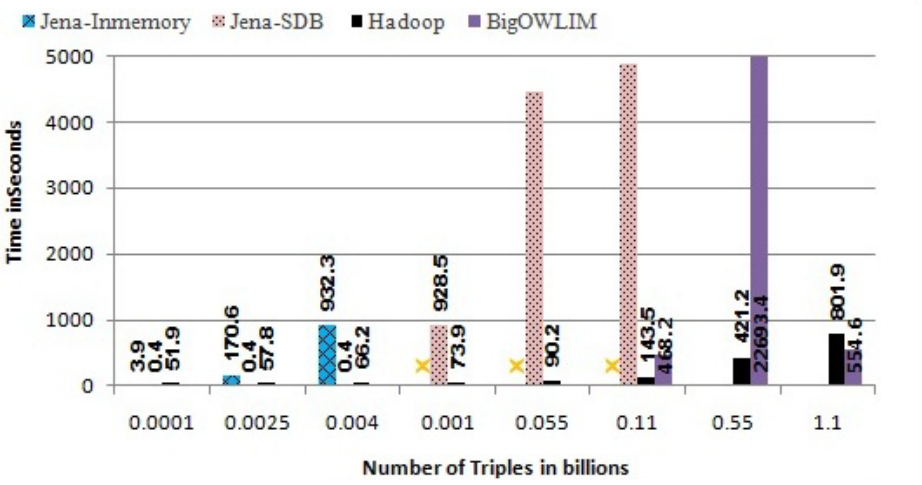
Multiple combinations possible

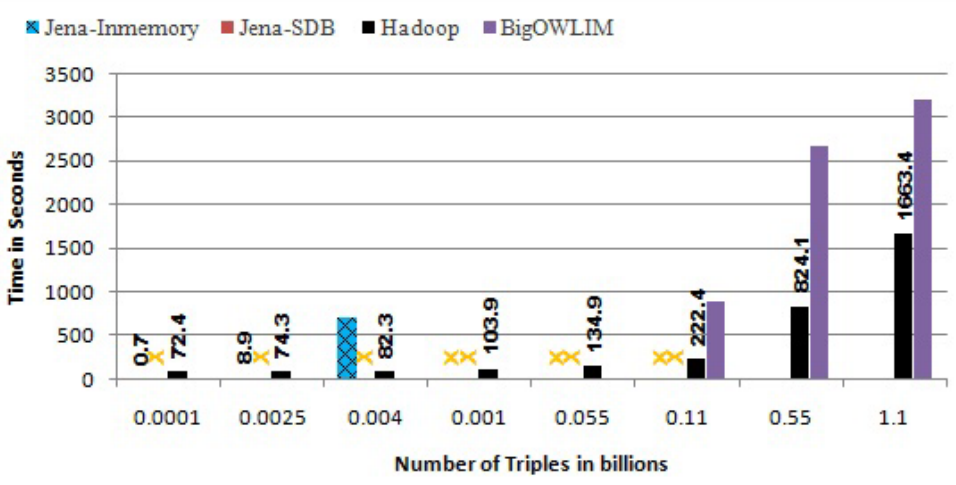
Constraint: Two adjacent nodes of Join Graph cannot be WHITE at the same time.



For each, a different plan will be created.

Some Numbers - Conclusion





- The framework performs much better than the other frameworks on the market.
- The framework is highly scalable, just add new nodes.
- When new data is added to the dataset, the time needed to answer the query does not increase as much as data size.

Alternative Model needed - Why ?

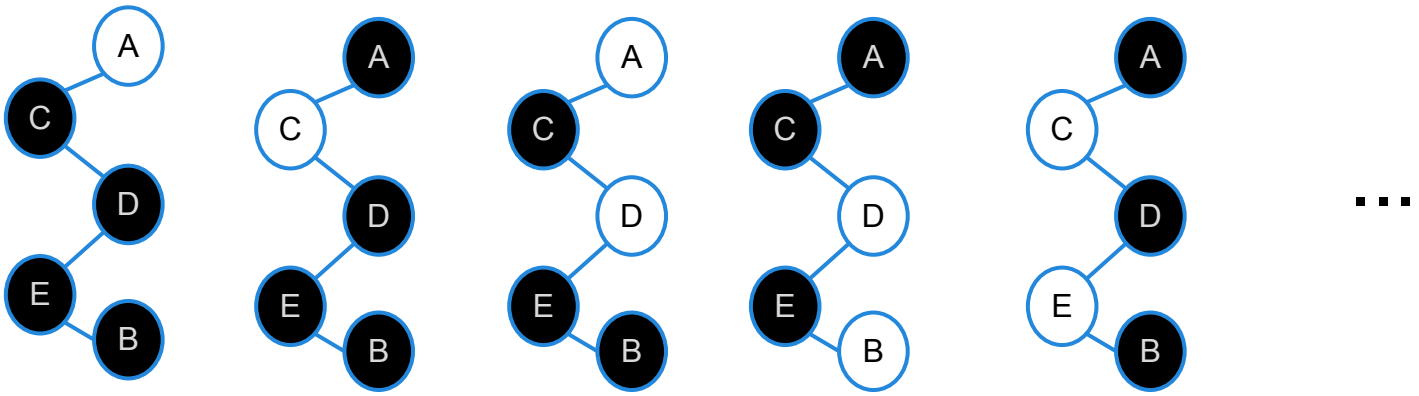
Ideal model is not very efficient:

- Simple abstract costs

  Ignores costs like copying, sorting, overhead of running jobs in Hadoop.

  Disk operations - Network transfers are expensive. A query plan with less intermediate data can perform worse than a plan with less jobs.

- Compherensive summary statistics



Alternative Model needed - Why ?

- Compherensive summary statistics

Example:

3 predicates: p1, p2, p3

Sub1	p1	?ObjV
?SubV1	p2	Obj1
?SubV2	p3	Obj2

3 predicates → 2^3 types of joins possible

p predicates → 2^p type of joins

Tens - hundreds of predicates ?

Relaxed Bestplan

Create a query plan with minimum possible jobs.

Joining Variable: A variable that is common in two or more triple patterns.

Ex: SELECT ?Y, ?V WHERE

{

?Y rdf:type ub:University

?Y rdf:name ?V

}

Y is a join variable, V is not.

Complete Elimination:

A join operation that eliminates a joining variable.

Ex: Job = Join (XY,Y) eliminates Y completely.

Partial Elimination:

A join operation that partially eliminates a joining variable.

Ex: Job = Join(XY,Y) & Join(X,ZX)

Resultant TP: {X, Z} → X is partially eliminated.

Relaxed Bestplan

E-count(v): The number of joining variables in the resultant triple pattern after a complete elimination of variable v.

Example:

Q = {X;Y;Z;XY;XZ}

For X : join(X, XY, XZ) : YZ is left E-count(x) = 2.

For Y: join (Y, XY) : X is left E-count(y) = 1.

For Z: join (Z, XZ) : X is left E-count(z) = 1.

How the algorithm works:

1- Remove non-joining variables from query.

2- Sort the variables according to the E-count.

3- For the next variable on E-count list, apply elimination.

4- Keep the resultant variable in a temporary set for next job.

5- Store the join.

6- Update Query Q. Move to the next variable. Go to step 3.

7- When Job is complete, run on the temporary set.

Relaxed Bestplan Example

1- Query Q = {X;Y;VZ;XY;XZ}. Remove the non-joining variables:

Q = {X;Y;Z;XY;XZ}

2- U = {Y;Z;X}, since Y and Z have E-count = 1, and X has E-count = 2.

3- U[0] = Y. Join(Y;XY).

Resultant Triple: X. Temp {X}

Q = {X;Y;Z;XY;XZ} - {Y;XY} = {X;Z;XZ}

4- U[1] = Z. Join(Z;XZ)

Resultant Triple: X. Temp {X, X}

Q = {X;Z;XZ} - {Z;ZX} = {X}

Only X is left. Job1 is done.

Job1 = {join(Y;XY), join(Z;XZ)}

Remaining: temp{X,X,X}

5- On the second job, we'll have a single join between remaining triples which is {X,X,X}

Job2 = join(X;X;X)

The Query Plan: {Job1,Job2} = {join(Y;XY), join(Z;XZ)}, join(X;X;X)

Summary statistics

Statistics are used for Breaking Ties

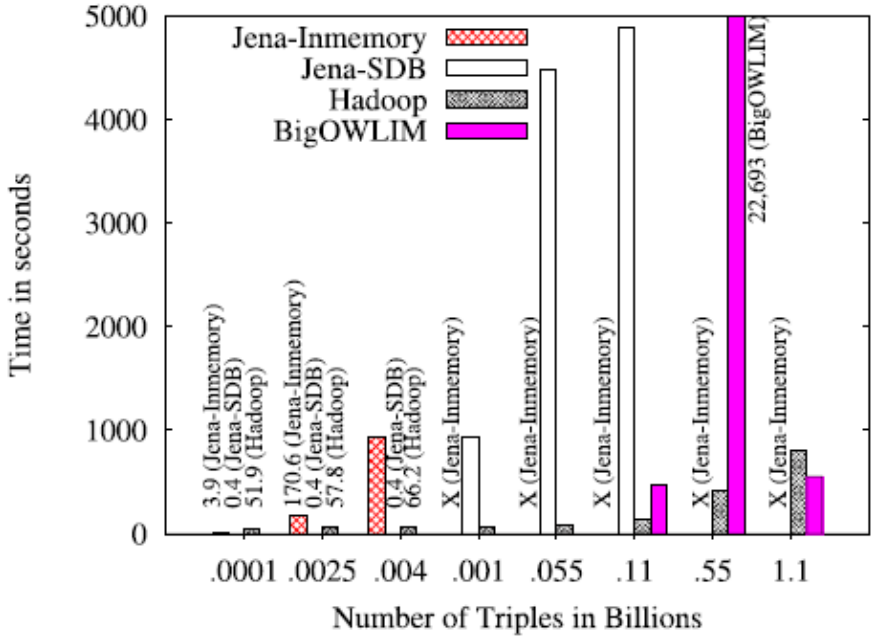
Example:

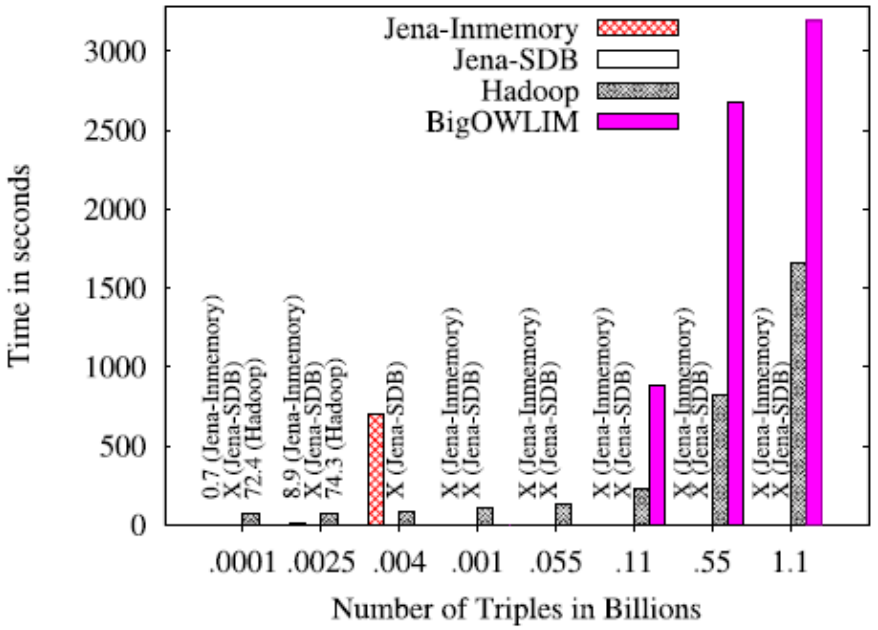
?X rdf:type ub:FullProfessor.
?X ub:advisorOf ?Y.
?Y rdf:type ub:ResearchAssistant.

We can generate 2 possible plans.
1- First two triples then last,
2- Last two triples then first.

To decide, we use statistics...

Some Numbers - Conclusion





- Relaxed Bestplan performs better than the solutions on the marked.
- Unlike the Framework introduced before, relaxed bestplan does not need comprehensive summary statistics.
- In case of slow network connection between nodes, the new framework will perform much better than the old one.

Thank You !