

Ge Song, Frédéric Magoulès (Supervisor), Fabrice Huet
(Co-Supervisor)
Lab MICS
CentraleSupélec
Université Paris-Saclay

Parallel and Continuous Join Processing for Data Stream

Présidente du Jury

M. Abdelkader Hameurlain, Professeur, IRIT, Rapporteur

M. Johan Montagnat, Charge de Recherche, CNRS, Rapporteur

M. Lei Yu, Professeur, École Centrale de Pékin, Examinateur

Mme. Bich-Lièn Doan, Professeur adjoint, CentraleSupélec, Examinateur

M. Frédéric Magoulès, Professeur, CentraleSupélec, Directeur de thèse

M. Fabrice Huet, Professeur, assistant, I3S, Co-Encadrant de thèse

Table of Contents

Introduction

Part I: Data Driven Stream Join (kNN)

Part II: Query Driven Stream Join (RDF)

Conclusion and Future Work

Introduction



Big Data Everywhere

- Google: 24 PB / day
- Facebook: 10 million photos + 3 billion “likes” / day
- Youtube: 800 million visitors / month
- Twitter: Doubling its size every year

Issues

- The most significant issue comes from the size of Big Data.
- The flip side of size is speed.
- Transfer cost.
- **Dynamic data – Data Stream**

Dynamic Data Stream

- Persistent **Static** Relations: **Batch-oriented** data processing
 - Transient **Dynamic** Data Streams: Real-time **stream** processing
-
- **Architecture Level:** add or remove computational nodes based on the current load
 - **Application Level:** withdraw old results and take new data into account

Objective: parallel and continuous processing for Join operation

Join: a popular and often used operation in the big data area.

- Data Driven Join : kNN (Data Parallelism)
- Query Driven Join : Semantic Join on RDF data (Task Parallelism)

Part I: Data Driven Stream Join (kNN)



Outline

- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Outline

- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Introduction

Definition: kNN

Given a set of query points R and a set of reference points S , a k **nearest neighbor join** is an operation which, for each point in R , discovers the k nearest neighbors in S .

- Query never changes
- Data changes: GPS (2 Dimensions), Twitter (77 Dimensions), Images (128 Dimensions) etc.

Introduction: Basic Idea

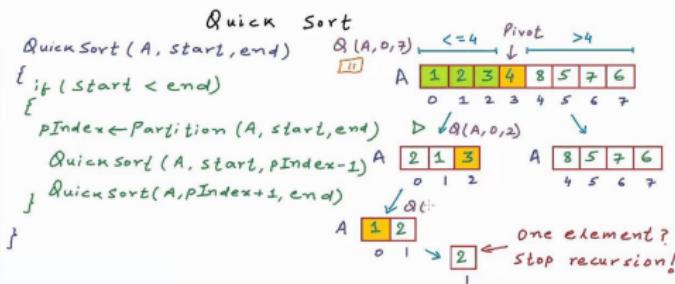
- Nested Loop – Calculate the Distances (Complexity $O(n^2)$)

```

for(int r : R){
    for(int s : S){
        Distance(r, s);
    }
}

```

- Sort – Find the top k smallest distance for each element
(Complexity: Quick Sort: $n \cdot \log(n)$, Priority Queue: $\log(n)$)



Outline

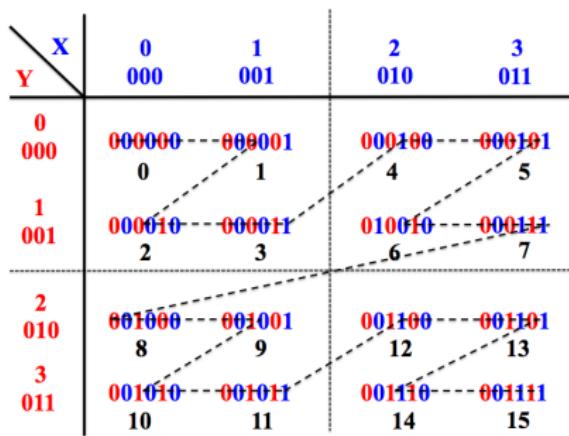
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Parallel Workflow

- Data Preprocessing
 - To reduce the dimension of data
 - To select central points of data clusters
- Data Partitioning
 - Distance Based Partitioning Strategy
 - Size Based Partitioning Strategy
- Computation
 - One Round Job
 - Two Rounds jobs

Data Preprocessing – Reducing the dimension of data

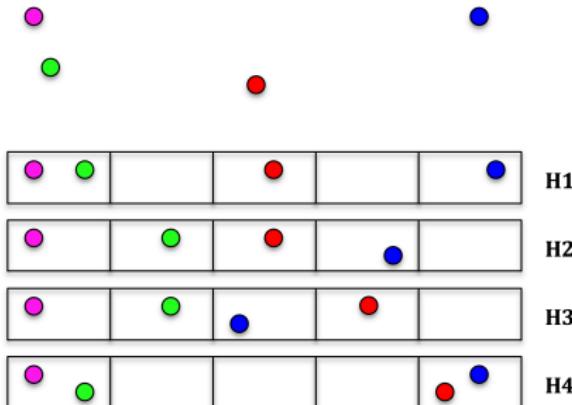
Space Filling Curve (Z-Value) – Project high dimensional data to 1 dimension while preserving the locality information



[z-value]: Efficient parallel kNN joins for large data in MapReduce, EDBT 2012, Chi Zhang et. al.

Data Preprocessing – Reducing the dimension of data

Locality Sensitive Hashing (LSH) – Map the neighbor points into the same buckets with a high probability using Hash Families

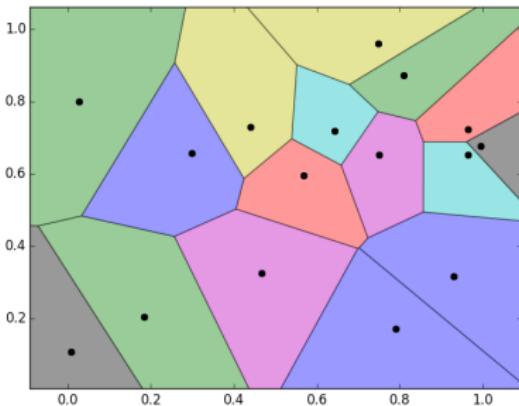


[LSH]: RankReduce - processing K-Nearest Neighbor queries on top of MapReduce, LSDS-IR 2010, Aleksandar Stupar et. al.

Data Preprocessing – Selecting central points (Pivots) of data clusters

Voronoi Diagram:

- Random Selection
- Furthest Selection
- K-Means Selection

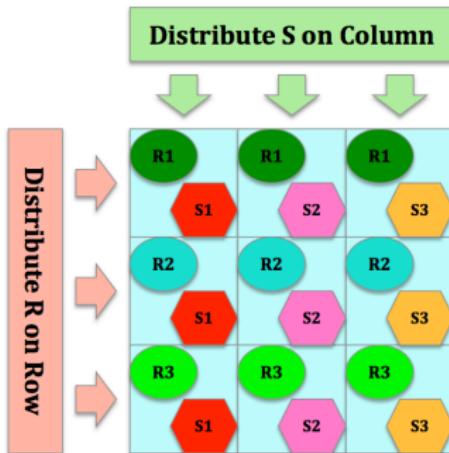


[voronoi]: Efficient Processing of k Nearest Neighbor Joins using MapReduce, VLDB 2012, Wei Lu et. al.

Parallel Workflow

- Data Preprocessing
 - To reduce the dimension of data
 - To select central points of data clusters
- Data Partitioning
 - Distance Based Partitioning Strategy
 - Size Based Partitioning Strategy
- Computation
 - One Round Job
 - Two Rounds jobs

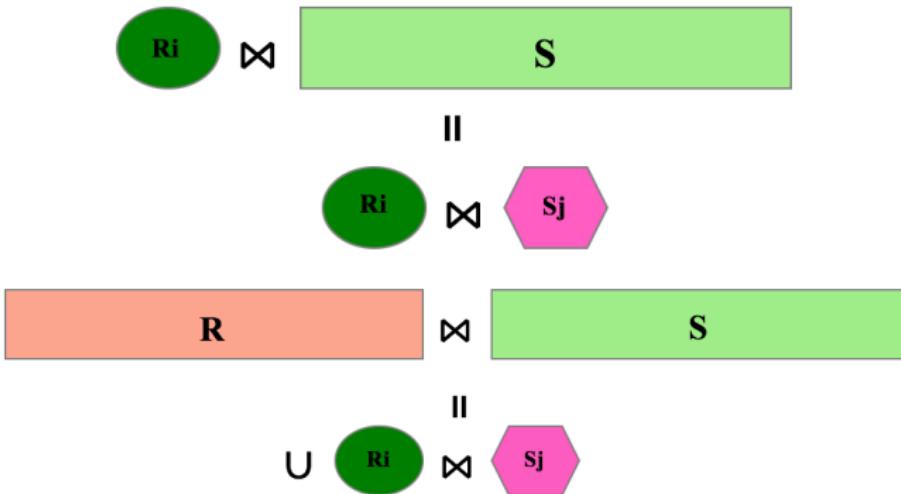
Data Partitioning – Basic Idea (Block Nested Loop)



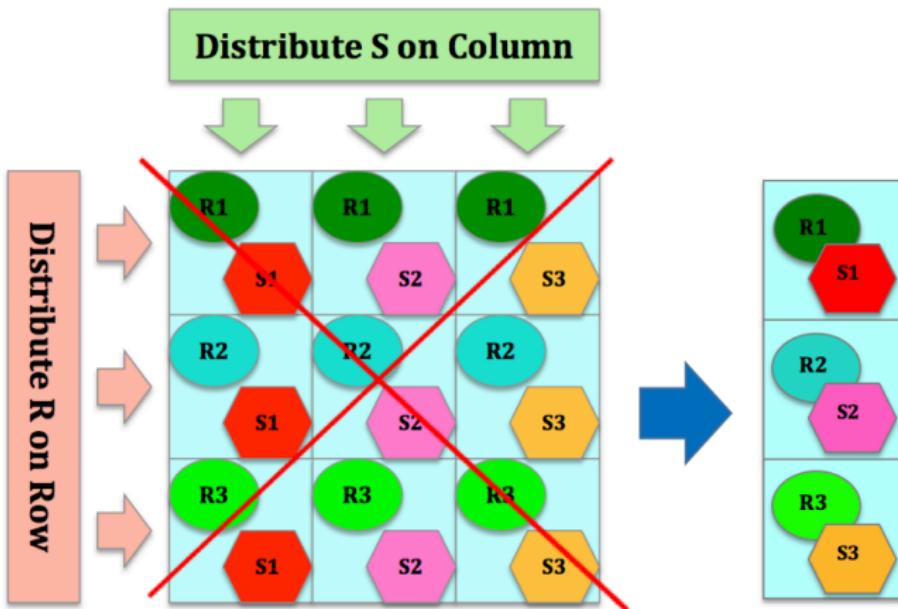
Problem: n^2 tasks for calculating pairwise distances; wastes a lot of hardware resources, and ultimately leads to low efficiency.

Data Partitioning – Motivation

The key to improve the performance is to preserve spatial locality of objects when decomposing data for tasks.



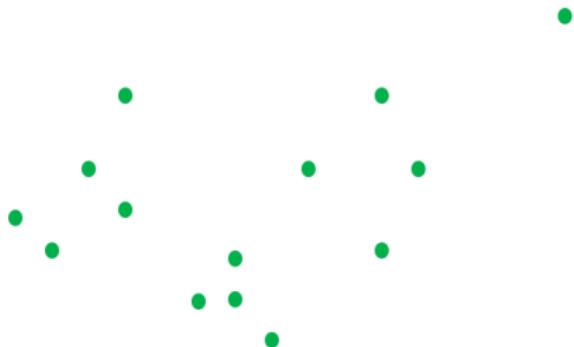
Data Partitioning – Motivation



Data Partitioning – Distance Based Partitioning Strategy – Voronoi Diagram

This strategy wants to have the most relevant points in each partition.

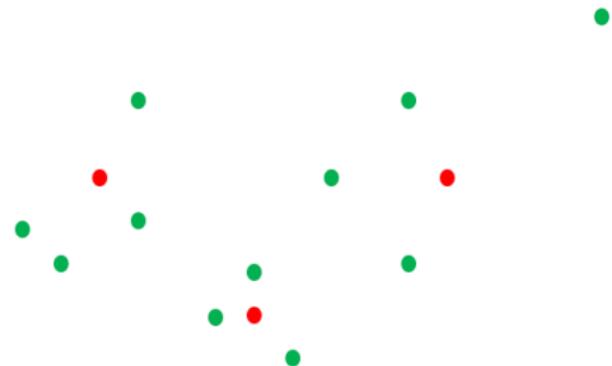
- 1 Selection Pivots in R
- 2 Partition R
- 3 Upper Bound in R
- 4 Find corresponding partition in S for each R



Data Partitioning – Distance Based Partitioning Strategy – Voronoi Diagram

This strategy wants to have the most relevant points in each partition.

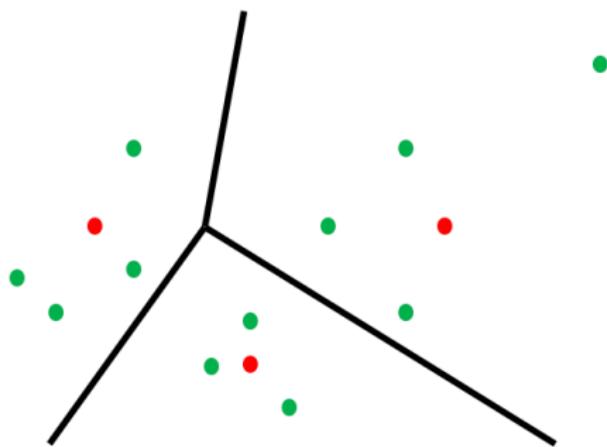
- 1 Selection Pivots in R
- 2 Partition R
- 3 Upper Bound in R
- 4 Find corresponding partition in S for each R



Data Partitioning – Distance Based Partitioning Strategy – Voronoi Diagram

This strategy wants to have the most relevant points in each partition.

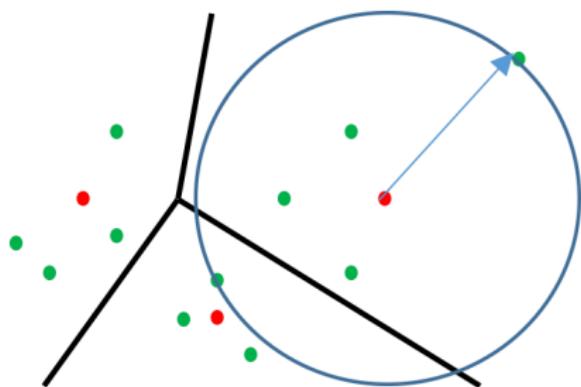
- 1 Selection Pivots in R
- 2 Partition R
- 3 Upper Bound in R
- 4 Find corresponding partition in S for each R



Data Partitioning – Distance Based Partitioning Strategy – Voronoi Diagram

This strategy wants to have the most relevant points in each partition.

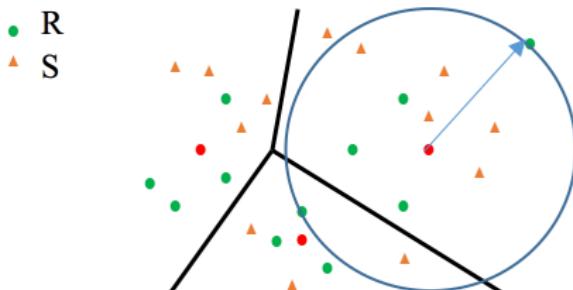
- 1 Selection Pivots in R
- 2 Partition R
- 3 Upper Bound in R
- 4 Find corresponding partition in S for each R



Data Partitioning – Distance Based Partitioning Strategy – Voronoi Diagram

This strategy wants to have the most relevant points in each partition.

- 1 Selection Pivots in R
- 2 Partition R
- 3 Upper Bound in R
- 4 Find corresponding partition in S for each R



Data Partitioning – Size Based Partitioning Strategy – Z-Value

This strategy wants to make each partition have equal size in order to achieve a good load balance.

- 1 A Sample of R
- 2 Find Quantiles in the Sample as the Bounds of the Partitions in R
- 3 Find corresponding partition in S for each R



Data Partitioning – Size Based Partitioning Strategy – Z-Value

This strategy wants to make each partition have equal size in order to achieve a good load balance.

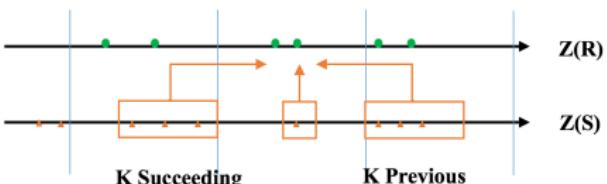
- 1 A Sample of R
- 2 Find Quantiles in the Sample as the Bounds of the Partitions in R
- 3 Find corresponding partition in S for each R



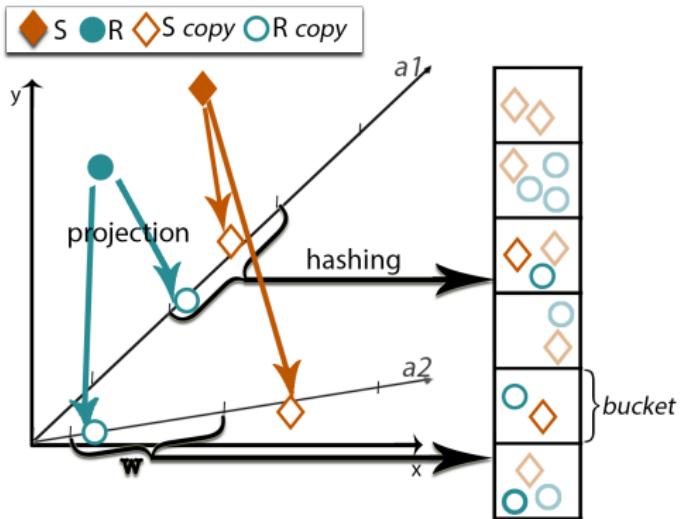
Data Partitioning – Size Based Partitioning Strategy – Z-Value

This strategy wants to make each partition have equal size in order to achieve a good load balance.

- 1 A Sample of R
- 2 Find Quantiles in the Sample as the Bounds of the Partitions in R
- 3 Find corresponding partition in S for each R



Data Partitioning – Size Based Partitioning Strategy – LSH



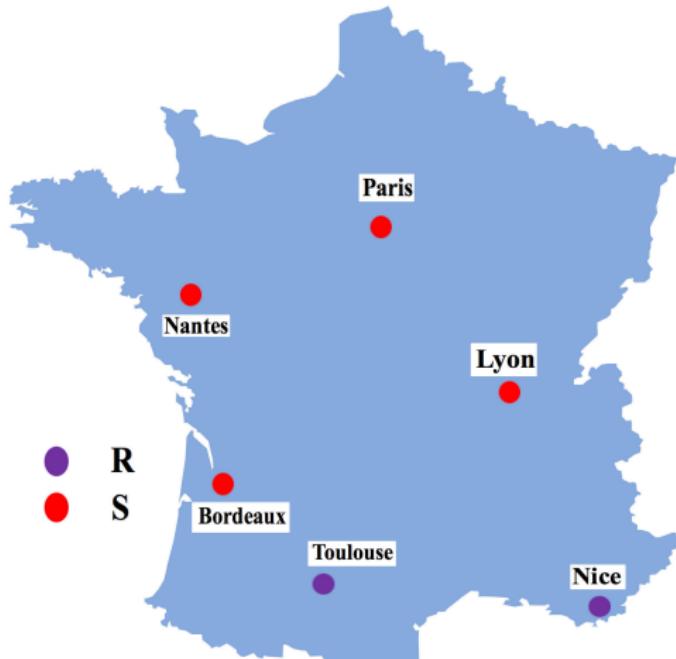
Computation

- One job – Directly give the global results
- Two consecutive jobs – First give the local results, then merge the local results into the global results

Purpose: using multiple rounds of jobs in order to reduce the number of elements to be sorted.

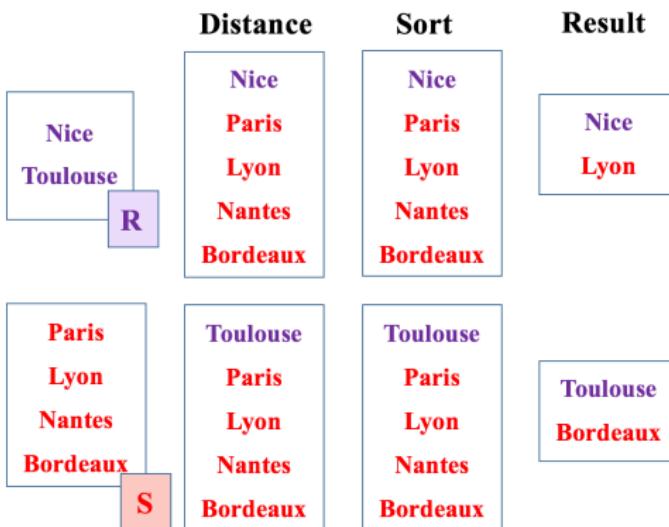
Computation – Example

For each city in R, find the nearest city in S.



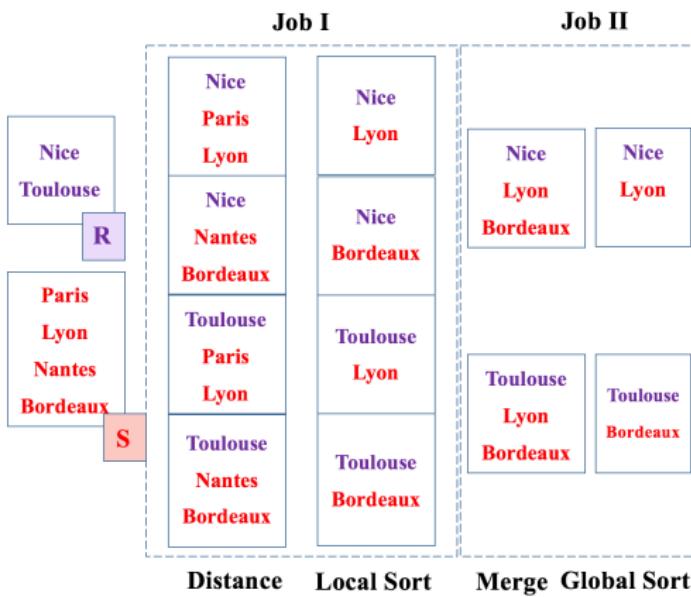
Computation – Example

One Job:

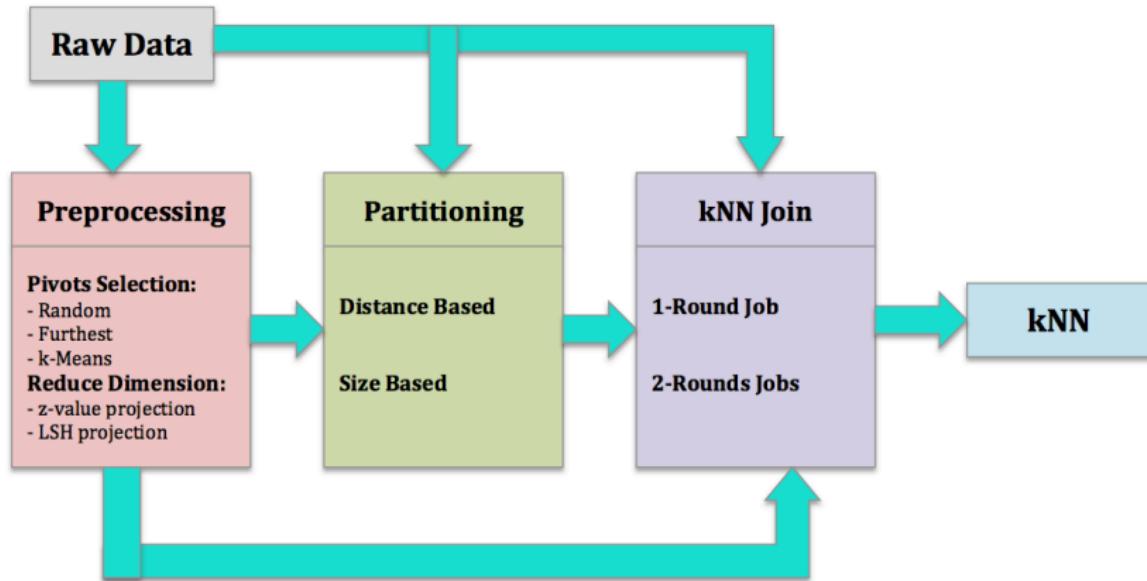


Computation – Example

Two Jobs:



Workflow



Outline

- Introduction
- Parallel Workflow
- Theoretical Analysis
 - Load Balance
 - Accuracy
 - Complexity
- Continuous kNN
- Experiment Result
- Conclusion

Load Balance

What is Load Balance?

$$|R_i| \times |S_i| = |R_j| \times |S_j|$$

Sub-Optimal Option

$\forall i \neq j,$
if $|R_i| = |R_j|$ or $|S_i| = |S_j|$,
then $|R_i| \times |S_i| \approx |R_j| \times |S_j|$

Load Balance

if $|R_i| = |R_j|$, the Worst Case Complexity is:

$$\mathcal{O}(|R_i| \times \log |S_i|) = \mathcal{O}\left(\frac{|R|}{p} \times \log |S|\right)$$

if $|S_i| = |S_j|$, the Worst Case Complexity is:

$$\mathcal{O}(|R_i| \times \log |S_i|) = \mathcal{O}\left(|R| \times \log \frac{|S|}{p}\right)$$

$p \ll |S| \Rightarrow |R_i| = |R_j|$ is better.

All advanced partitioning strategies first partition R into equal sized partitions, then find the corresponding S for each R.

Accuracy

The lack of accuracy is the direct consequence of techniques to reduce the dimensionality with techniques of as z-values and LSH.

- **Z-Value**

- Depends on k
- Shift of data – move data in the direction of a random vector
- Increase the number of shifts of data will decrease the error rate

- **LSH**

- Depends on parameter tuning
- Increase the number of hash functions will decrease the error rate

Complexity

- **The number of MapReduce jobs:** starting a job requires some initial steps.
- **The number of Map tasks and Reduce tasks used to calculate $k\text{NN}(R_i \times S)$:** the larger this number is, the more information is exchanged through the network.
- **The number of final candidates for each object r_i :** Finding the top k results is very time consuming.

Main Overhead

- Communication overhead:
 - the amount of data transmitted over the network
- Computation overhead:
 - computing the distances
 - finding the k smallest distances

Outline

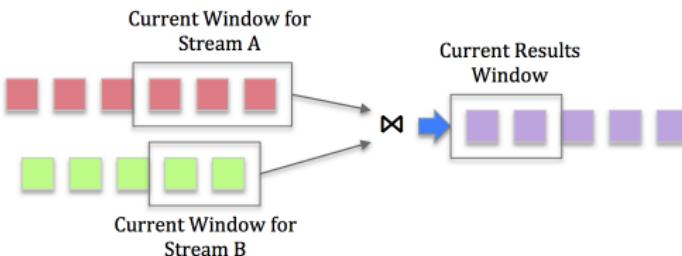
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Sliding Window Model – Motivation

- Unbounded streams can not be wholly stored in bounded memory
- New items in a stream are more relevant than older ones.

Sliding Window Model

Maintaining a moving window of the most recent elements in the stream



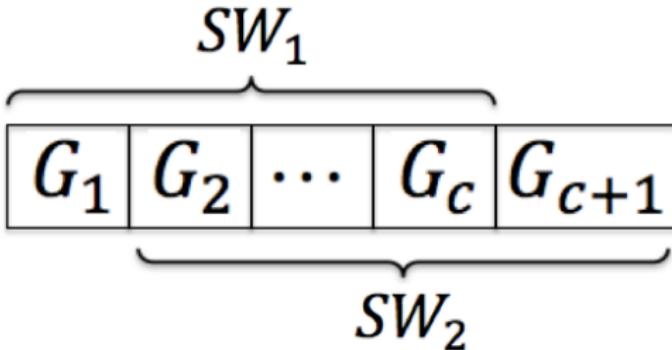
Sliding Window – Two Strategies

- **Re-Execution Strategy**
 - **Eager Re-execution Strategies** – Generating new results right after each new data arrives
 - **Lazy Re-execution Strategies** – Re-Executing the query periodically
- **Data Invalidation Strategy**
 - **Eager Invalidation Strategies** – Scanning and moving forward the sliding window upon arrival of new data
 - **Lazy Re-execution Strategies** – Removing old data periodically and require more memory to store data waiting for expiration

Sliding Window – Two Strategies

- **Re-Execution Strategy**
 - Eager Re-execution Strategies
 - **Lazy Re-execution Strategies**
- **Data Invalidation Strategy**
 - Eager Expiration Strategies
 - **Lazy Invalidation Strategies**

Re-Execution and Expiration Period – Generation



Different types of dynamic kNN joins

- Static R and Dynamic S (SRDS)
 - Exists rarely in real applications.
 - Reuse the parallel methods
- Dynamic R and Static S (DRSS)
 - Most used scenario in real applications
 - Reuse Random Partition method
- Dynamic R and Dynamic S (DRDS)
 - General situation
 - Basic Method + Advanced Method

Dynamic R and Dynamic S – Basic Method (Sliding Block Nested Loop)

n^2 tasks for each generation

S in i^{th} Generation				
R in i^{th} Generation	S_1	S_2	...	S_n
R_1	$G_i(R_1, S_1)$	$G_i(R_1, S_2)$...	$G_i(R_1, S_n)$
R_2	$G_i(R_2, S_1)$	$G_i(R_2, S_2)$...	$G_i(R_2, S_n)$
...
R_n	$G_i(R_n, S_1)$	$G_i(R_n, S_2)$...	$G_i(R_n, S_n)$

Dynamic R and Dynamic S – Advanced Method (Naive Bayes Partitioning)

n tasks for each generation: partition the new data items without moving the old ones.

Naive Bayes Theory

Given two independent events A and x, the conditional probability of given x and A occurs is:

$$P(A|x) = \frac{P(x|A) \cdot P(A)}{P(x)} \quad (1)$$

The new data x will be partitioned to the partition with the highest conditional probability.

Outline

- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Experiment Setting

Cluster Setting

- The experiments were run on two clusters of Grid'5000 with Hadoop 1.3
- The number of replications for each split of data is set to 3
- The number of slots of each node is 1

Datasets

- **OpenStreetMap** Geo dataset contains geographic XML data in two dimensions – Low Dimension
- **Catech 101** It is a public set of images, which contains 101 categories of pictures of different objects. (Speeded Up Robust Features – 128 dimensions) – High Dimension

Methods Evaluated

- **H-BkNNJ** Naive Method – Without preprocessing and partitioning – One Job
- **H-BNLJ** Block Nested Loop – Without preprocessing and partitioning – Two Jobs
- **PGBJ** Based on Voronoi – Preprocessing: Select Pivots – Distance Based Partitioning – One Job
- **H-zkNNJ** Based on Z-Value – Preprocessing: z-value – Size Based Partitioning – Two Jobs
- **RankReduce** Based on LSH – Preprocessing: LSH – Size Based Partitioning – Two Jobs

Evaluations

Impacts: (x axis)

- Impact of input data size
- Impact of k
- Impact of Dimension and Dataset

Measures: (y axis)

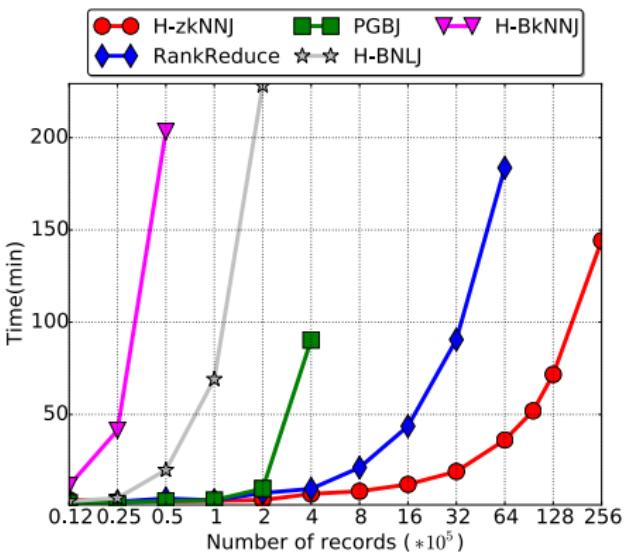
- Execution Time
- Communication Overhead
- Recall and Precision

Practical Analysis

Evaluation Result – Verify the theoretical Analysis

Execution Time for Geo dataset (2 dimensions):

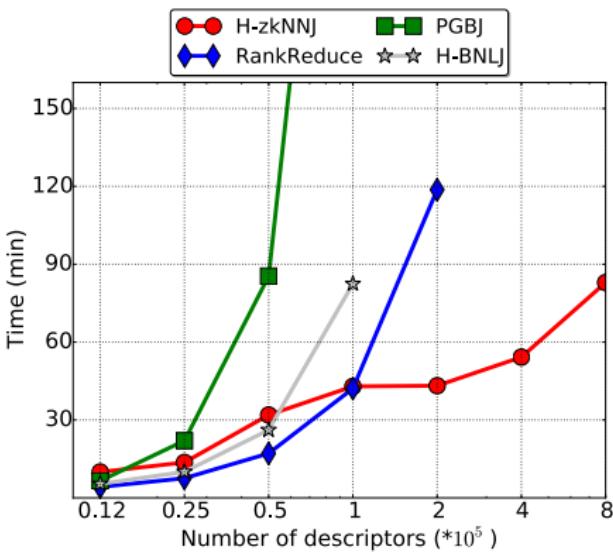
- H-BkNNJ: Naive
- H-BNLJ: Block Nested Loop
- PGBJ: Voronoi
- H-zkNNJ: z-value
- RankReduce: LSH



Evaluation Result – Surprise

Execution Time for Image dataset (128 dimensions):

- H-BkNNJ: Naive
- H-BNLJ: Block Nested Loop
- PGBJ: Voronoi
- H-zkNNJ: z-value
- RankReduce: LSH



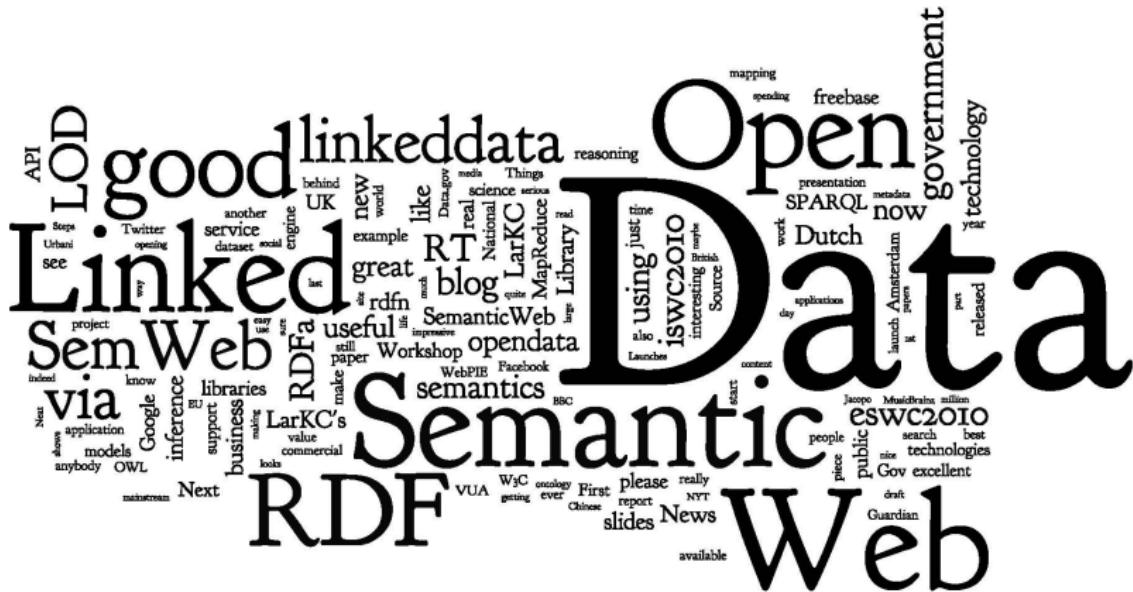
Outline

- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Conclusion

- Parallel Workflow for kNN Join
- Continuous kNN Join for Data Streams
- Theoretical Analysis
- Experimental Analysis for 5 Methods

Part II: Query Driven Stream Join (RDF)



Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Plan
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

Introduction – RDF Data Model

- **Resource Description Framework** is a data standard proposed by W3C
- It aims at representing semantic data in a machine understandable manner.
- This representation is used to describe semantic relations among data.
- Data items are expressed as triples in form of <subject, predicate, object> (e.g. <Sophie, hasSister, Ray>)
 - The subject of a triple indicates the resource that this triple is about.
 - The predicate refers to the property of the subject.
 - The object denotes to the projection value of the subject by the predicate.

Introduction – SPARQL Query Language

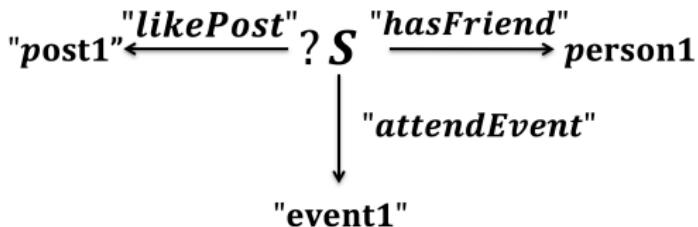
- SPARQL is a W3C recommendation query language for querying RDF data.
- The basic component of a SPARQL query is the triple patterns.
- A triple pattern is a special kind of triple where S, P and O can be either a literal or a variable.

An Example (Triple Pattern Representation):

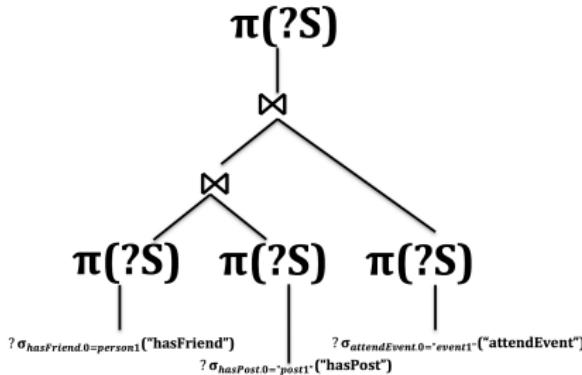
```
SELECT ?S
WHERE {
    Q1      ?S "hasFriend" person1 .
    Q2      ?S "likePost" "post1" .
    Q3      ?S "attendEvent" "event1"
}
```

Introduction – SPARQL Query Example

Graph Representation:



Relational Representation:

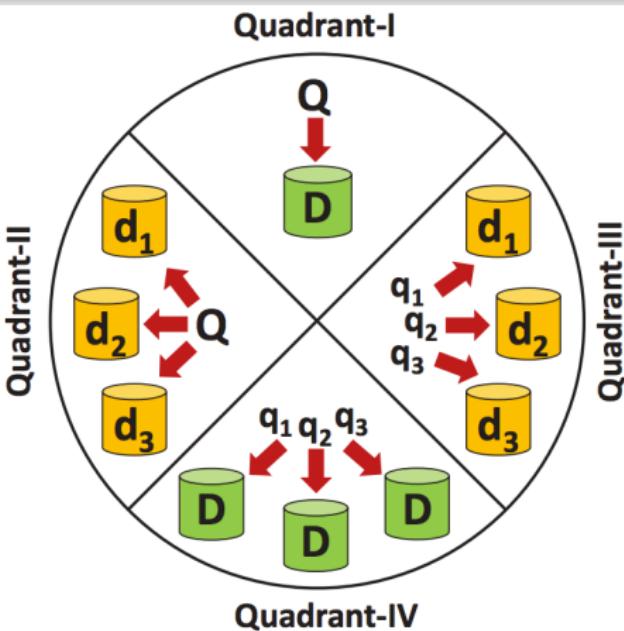


Query Driven Join

- Data format does not change
- Query changes

Related Works – 4 Types of Processing

- Q1: Centralize
- Q2 and Q4: Distribute either data or query
- Q3: Distribute both data and query (We use this manner)



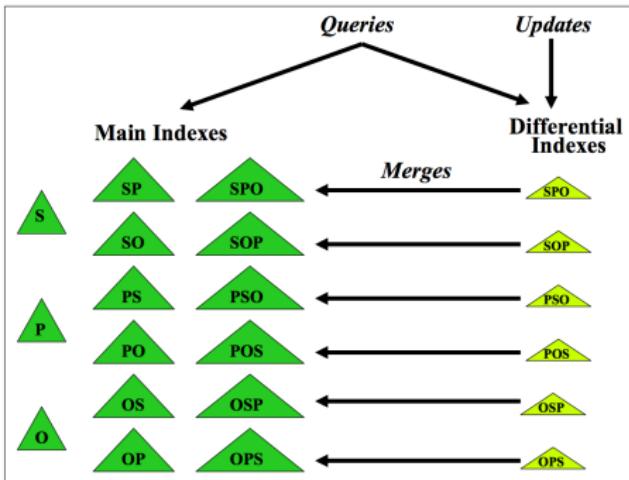
DREAM: distributed RDF engine with adaptive query planner and minimal communication, PVLDB 2015, Mohammad Hammoud et. al.

Related Work – Partitioning Strategies for RDF graphs

- Vertex Partitioning methods for graphs.
 - High overhead of loading big RDF graphs into the existing graph partitioner.
 - Requires the entire graph information in order to make decisions
 - Replication of the boundary of each partition in order to reduce the transmission of data
- Hash Partitioning based on indexes

Hash Partitioning – Index

Too many indexes!!!



RDF-3X: a risc-style engine for RDF, PVLDB 2008, Thomas Neumann et. al.

General Distributed Processing Steps

- Partition the RDF streams, and distribute these sub-streams to different nodes
- Decompose the queries into sub-queries and assign these sub-queries to the appropriate nodes
- Reply rapidly to the changes of data (the expiration of old data, and the update of new data), and return the results in real-time

Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

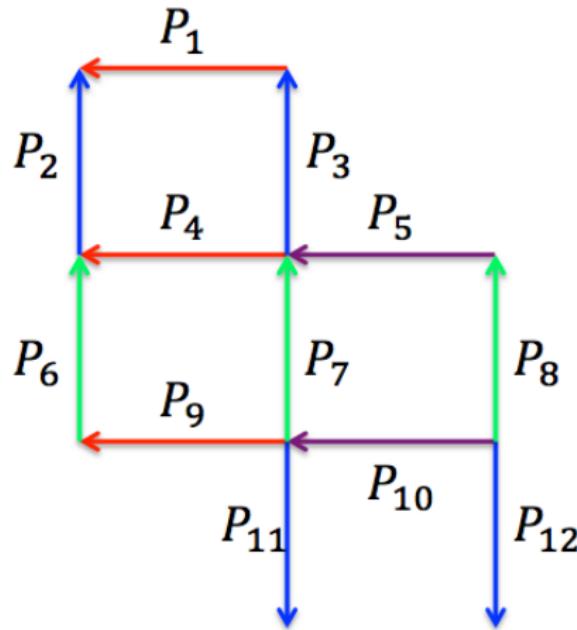
Query Decomposition

Decomposition Strategy: Divide the queries into triple patterns, send each triple pattern to different machines.

- It is simple, and does not require any complicated computations.
- The performance or accuracy of this method does not depend on the index or replication of data.
- The number of triple patterns is limited

Sub-Query Scheduling

Edge Coloring Method: Maximize Parallelism



Data Partitioning

Partitioning Strategy: The triples will be assigned to the nodes that hold the triple pattern with the same predicate.

- It does not require any index, and will not occupy more disk space.
- It can quickly adapt to changes in data, and it does not require any re-partitioning on the existing data when new data arrives or old data expires.

Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Analysis
- Implementation
- Experiment Result
- Conclusion

Challenges

- Communication among nodes
- Join of the intermediate results produced by each triple pattern
- Order of sending and receiving information

Communication among nodes – Bloom Filter

Do not send triples, send Bloom Filters



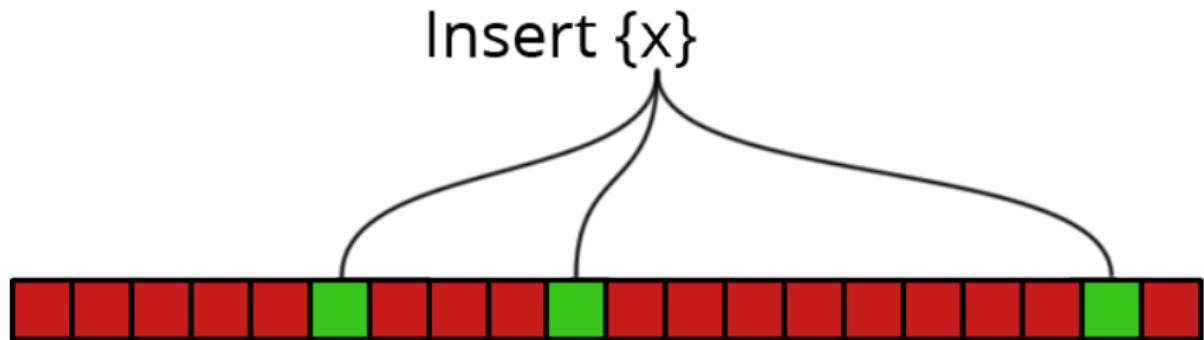
$$m = 20 \text{ bits}$$

$$k = 3 \text{ hash functions}$$

$$n = 3 \text{ insertions}$$

Communication among nodes – Bloom Filter

Do not send triples, send Bloom Filters



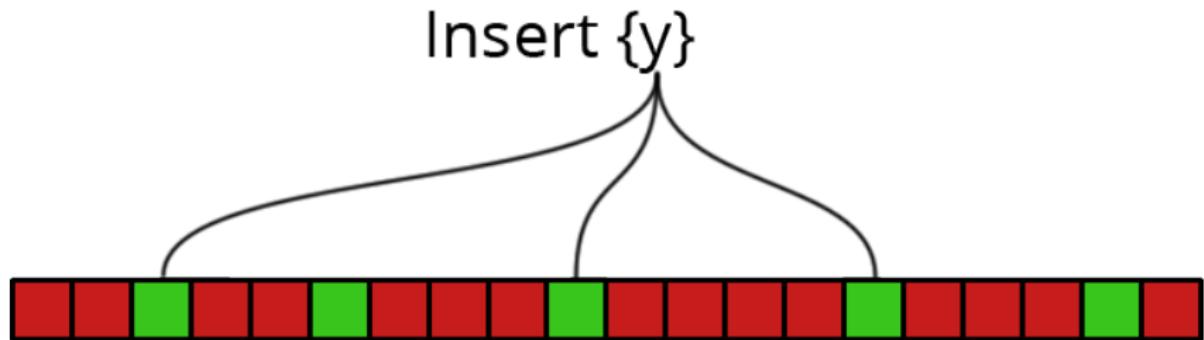
$$m = 20 \text{ bits}$$

$$k = 3 \text{ hash functions}$$

$$n = 3 \text{ insertions}$$

Communication among nodes – Bloom Filter

Do not send triples, send Bloom Filters



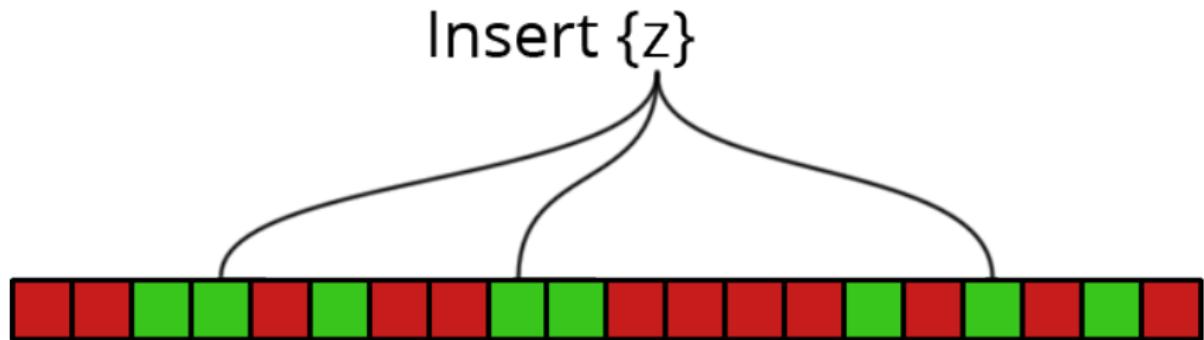
$$m = 20 \text{ bits}$$

$$k = 3 \text{ hash functions}$$

$$n = 3 \text{ insertions}$$

Communication among nodes – Bloom Filter

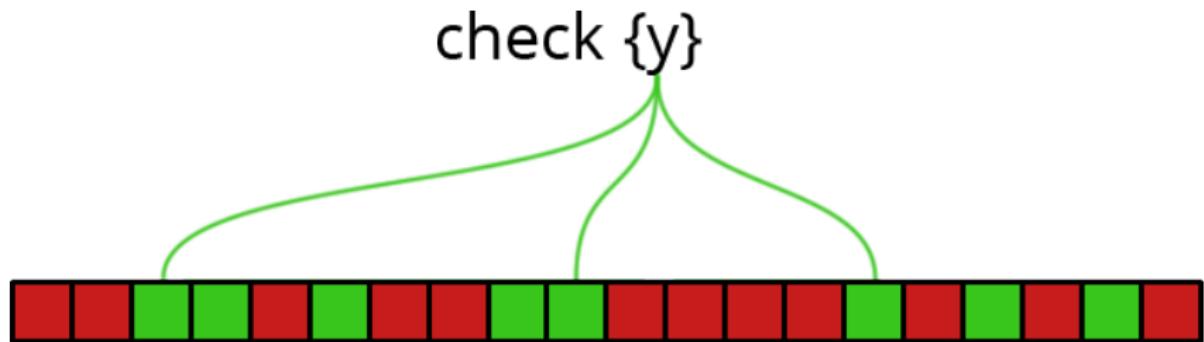
Do not send triples, send Bloom Filters



$m = 20$ bits
 $k = 3$ hash functions
 $n = 3$ insertions

Communication among nodes – Bloom Filter

Do not send triples, send Bloom Filters



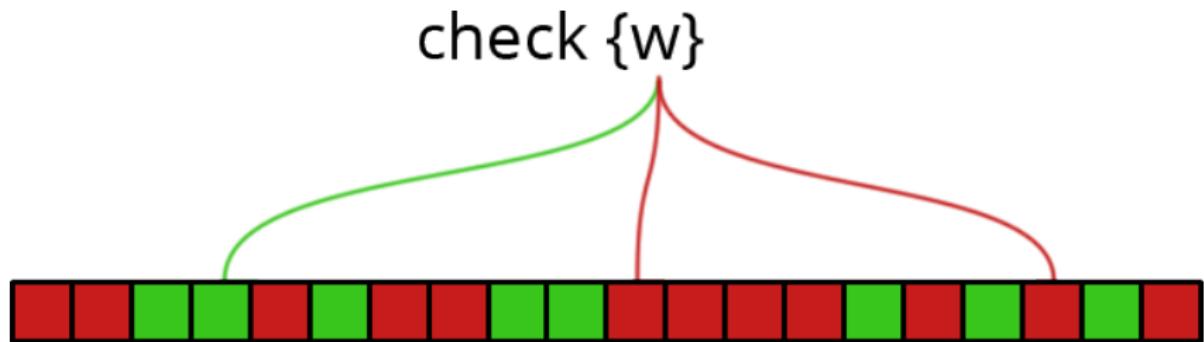
$$m = 20 \text{ bits}$$

$$k = 3 \text{ hash functions}$$

$$n = 3 \text{ insertions}$$

Communication among nodes – Bloom Filter

Do not send triples, send Bloom Filters



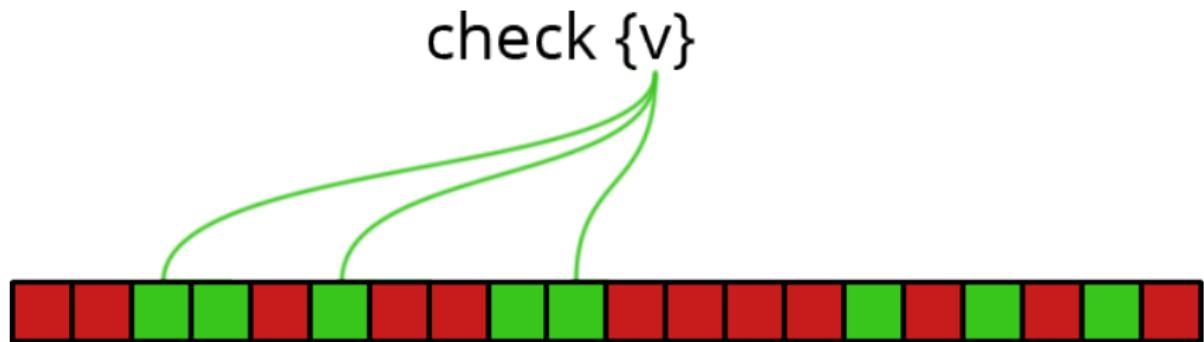
$m = 20$ bits

$k = 3$ hash functions

$n = 3$ insertions

Communication among nodes – Bloom Filter

Do not send triples, send Bloom Filters



$m = 20$ bits

$k = 3$ hash functions

$n = 3$ insertions

Communication among nodes – Bloom Filter

False Positive Rate

$$p = \left(1 - e^{-\frac{nk}{m}}\right)^k$$

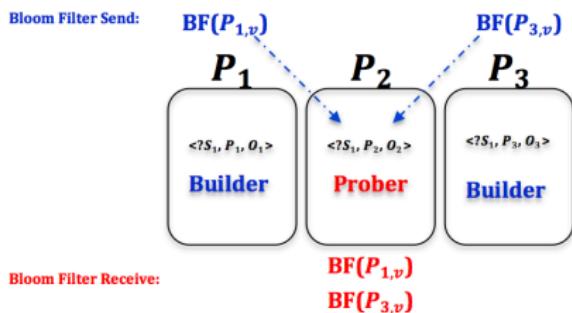
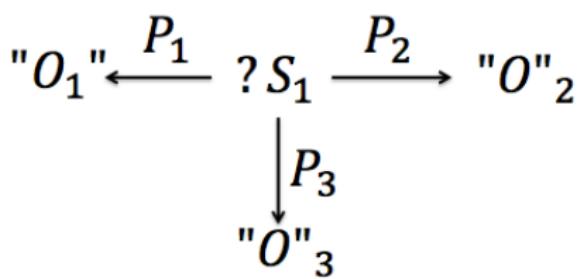
Bloom Filter – Build and Probe

- **Builder:** inserts the results of the triple pattern to the Bloom Filter.
- **Prober:** checks the existence of the results of the triple pattern in the Bloom Filter

Question: Which triple patterns should be **Builders**, and which ones should be **Probers**?

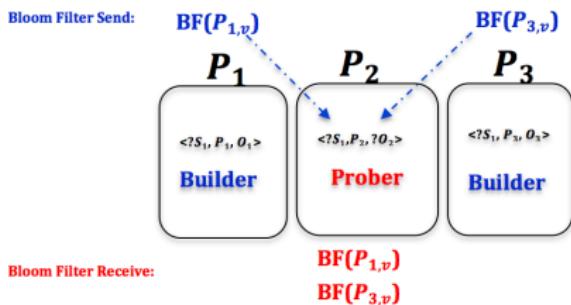
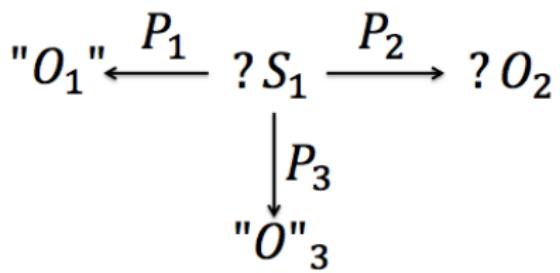
The join of the intermediate results – Structure Based Rules

Rule 1: 1-Variable Join



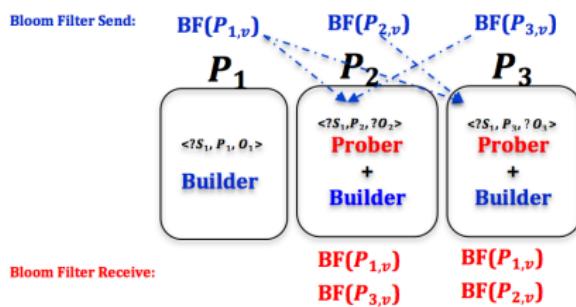
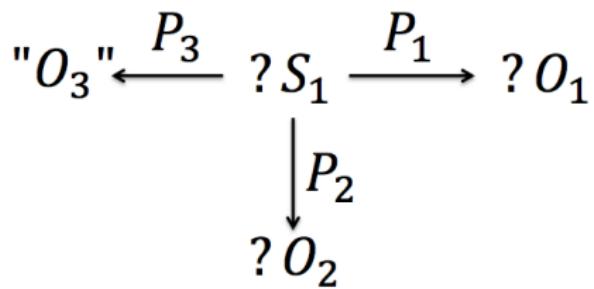
The join of the intermediate results – Structure Based Rules

Rule 2: 2-Variable Join



The join of the intermediate results – Structure Based Rules

Rule 3: Multiple-Variable Join



Question: what is the sending and receiving order?

The order of sending and receiving information

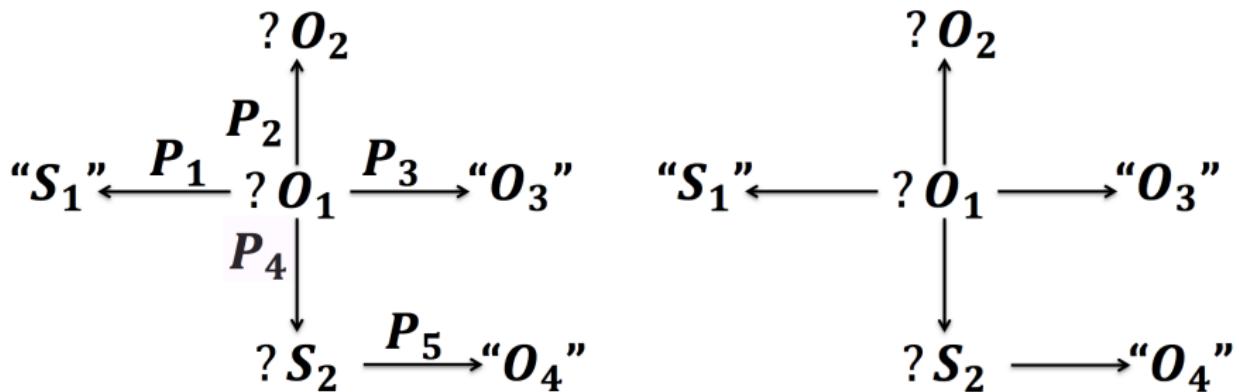
Rule 4: Query Topological Sort

Query Topological Sort

Query Topological Sort is a topological sort for the query graphs, where the constant nodes on the graph have higher priority than the variable nodes at the same level.

The order of sending and receiving information

Rule 4: Query Topological Sort

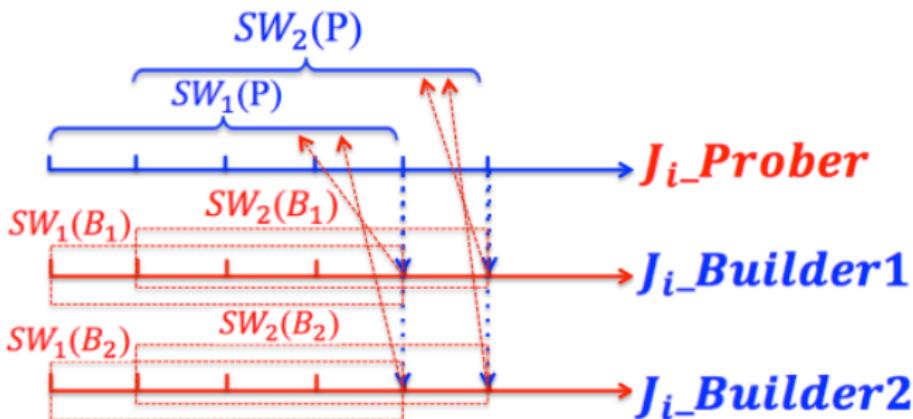


Results: $\{O_4, ?S_2, S_1, O_3, ?O_2, ?O_1\}$

Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

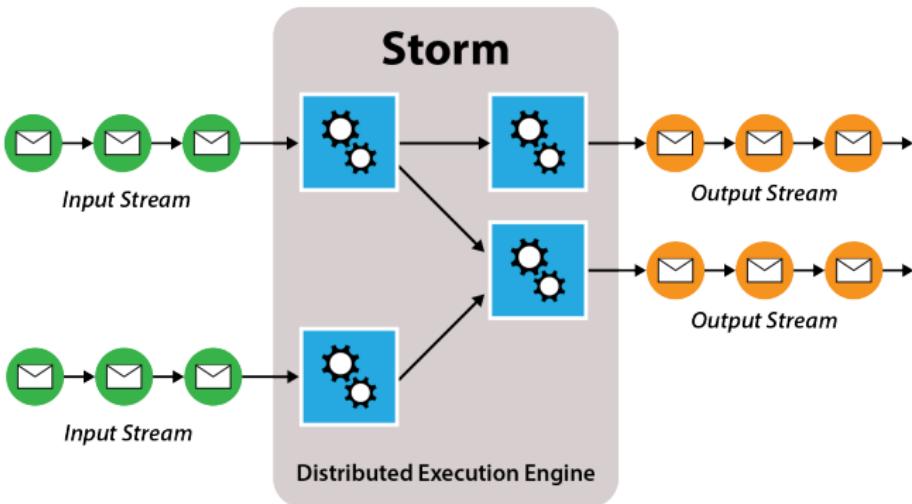
Continuous Join: Sliding Window + Sliding Bloom Filter



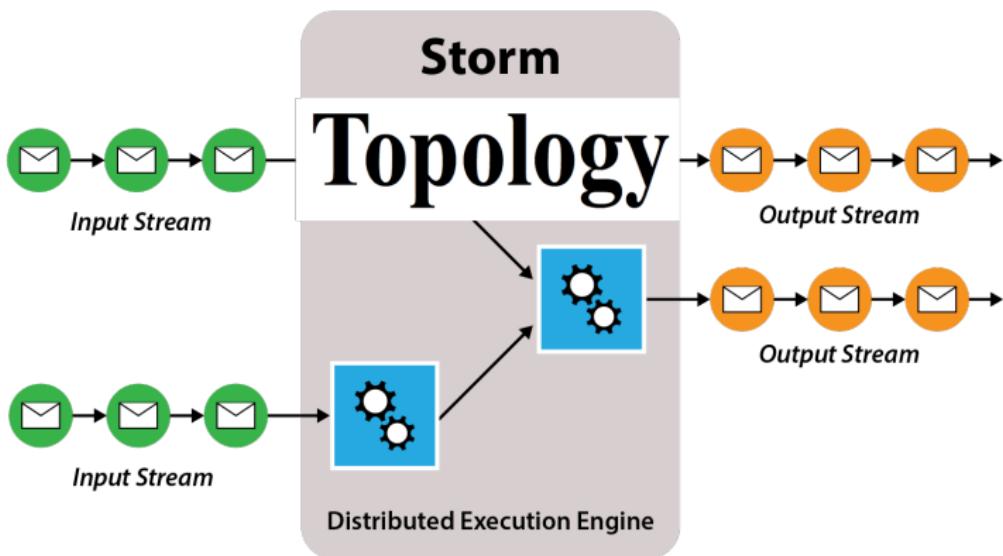
Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

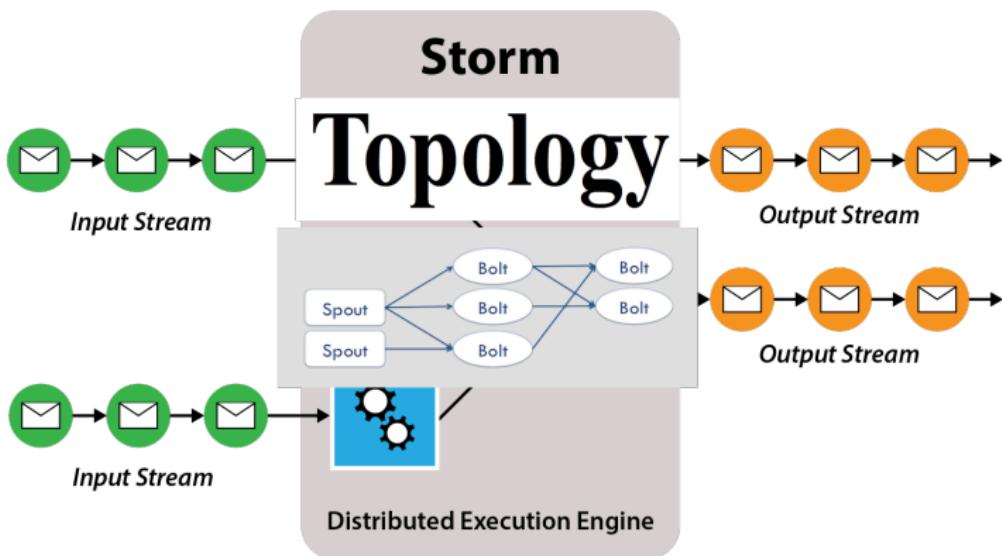
Apache Storm



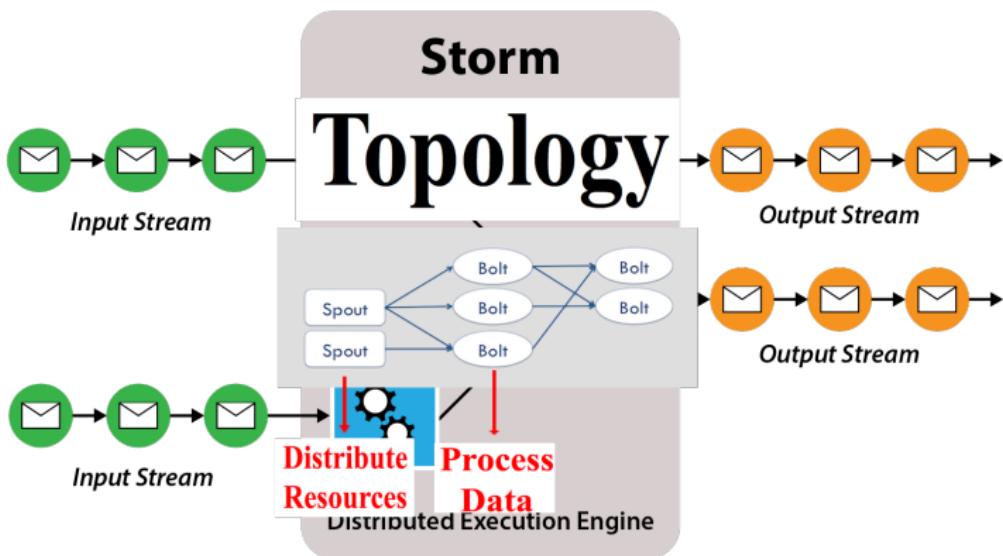
Apache Storm



Apache Storm

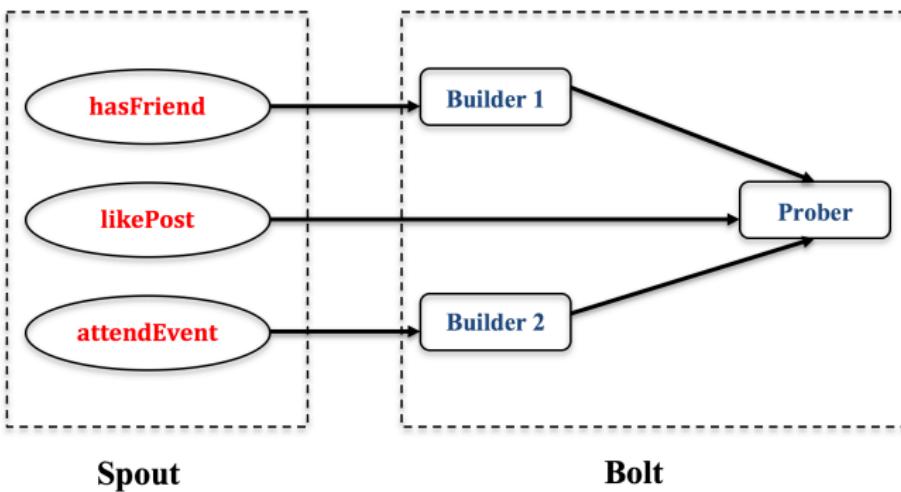


Apache Storm



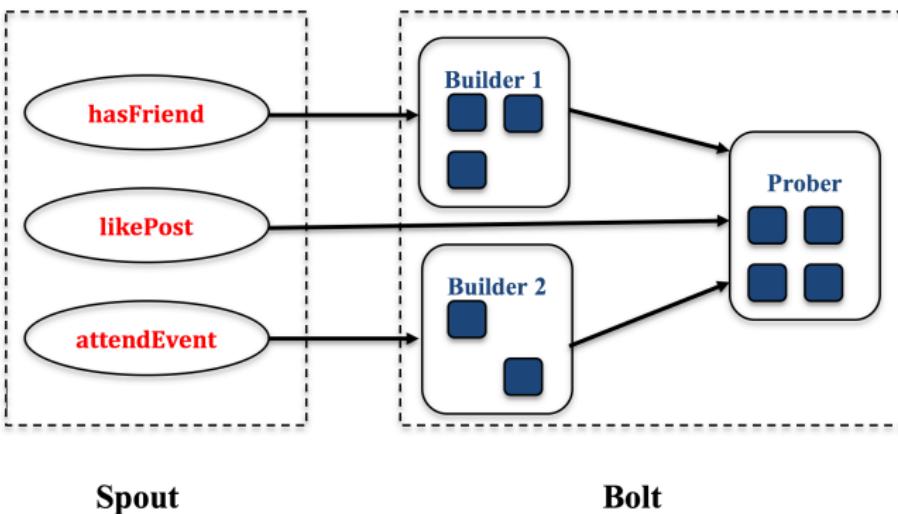
Implementation

```
SELECT ?S
WHERE{
    Q1    ?S "hasFriend" person1 .
    Q2    ?S "likePost" "post1" .
    Q3    ?S "attendEvent" "event1"
}
```



Implementation

```
SELECT ?S
WHERE{
    Q1    ?S "hasFriend" person1 .
    Q2    ?S "likePost" "post1" .
    Q3    ?S "attendEvent" "event1"
}
```



Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

Experiment Setting

- We evaluate the system on Grid 5000, with 11 nodes. 1 Master (Nimbus), 10 Slaves (Supervisor).
- The Storm version is 1.0, and we only use one slot on each machine.
- Apache Jena API is used for reading triples.

Data sets

- Synthetic data
 - The RDF triples generated in Spouts are distributed to the nodes according to their predicate.
 - Two sub-sets of data are generated. The first sub-set contains only the results of the join, and the second one contains only the non-results of the join.
- LUBM Data
 - LUBM has 14 queries, and we have tested query No.1, query No.3, and query No.4
 - It consists of a university domain.
 - It is customizable and repeatable.

Evaluations

Impacts: (x axis)

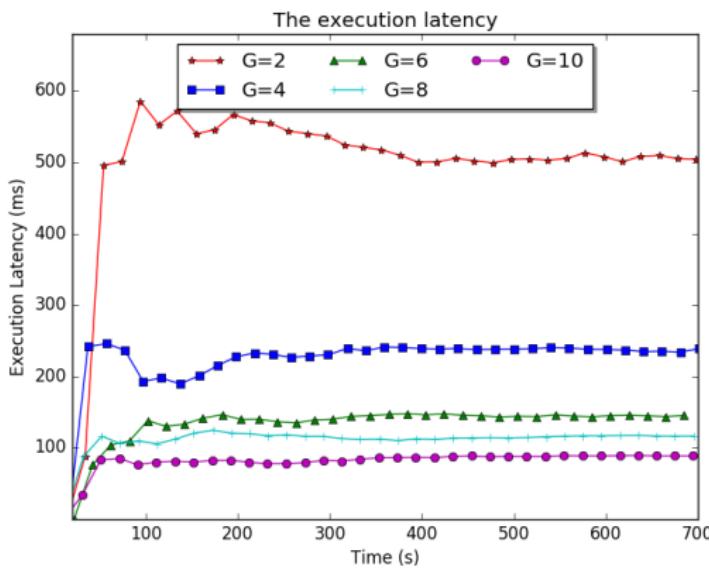
- Sliding Window Size
- Number of Generations

Metrics: (y axis)

- Execution Latency – The difference between the time an element is generated and the time it is emitted as a result
- Process Latency – The difference between the time an element is generated and the time it begins to be processed
- Data Transmitted
- Accuracy

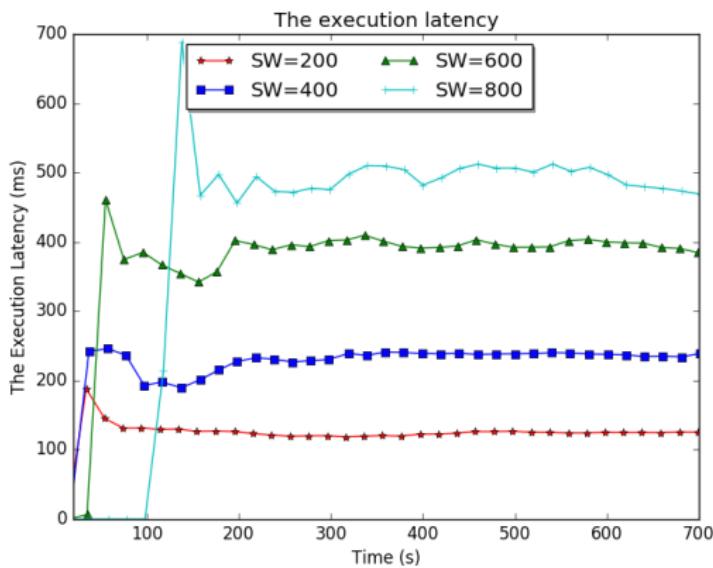
Execution Latency

Sliding Window Size = 800 (1V)



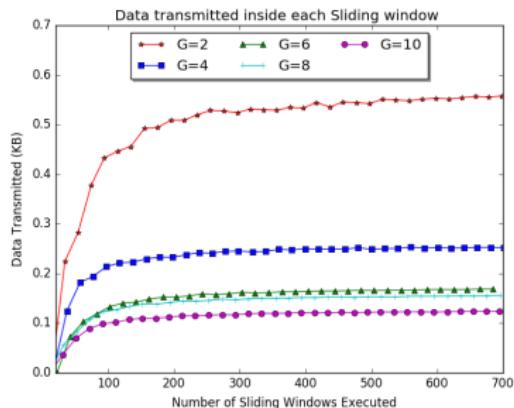
Execution Latency

Number of Generation = 6 (1V)

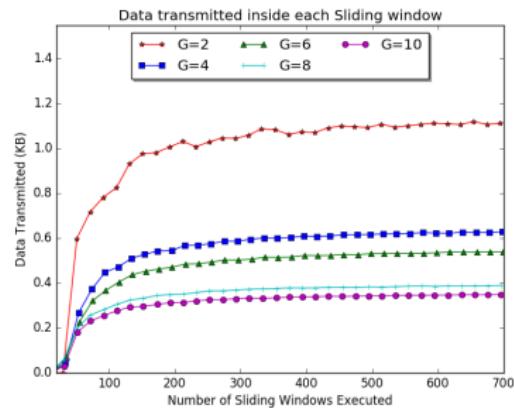


Data Transmitted – Sliding Window Size = 800

1V



MV



400 more data transmission without Bloom Filters.

Accuracy

We got 100% correct results – Surprise!

- use p (false positive rate) and n (number of elements to be inserted) to decide m (number of bits in the Bloom Filter)
- n is set to the number of elements received by the triple pattern
- the real number of elements inserted into the Bloom Filter should be the number of results which match the triple pattern (far less than n)

Outline

- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

Conclusion

- Parallel and distributed join processing on RDF streams
- Distribute both data and queries
- Efficient
 - Time
 - Execution Latency less than 600 ms
 - Process Latency less than 1 ms
 - Space – 400 times more efficient than without using Bloom Filters

Conclusion and Future Work



Conclusion

- Data Driven Stream Join
 - Data Preprocessing
 - Data Partitioning
 - Computation
- Query Driven Stream Join
 - Query Decomposition
 - Communication among Sub-Queries
 - Combine results of Sub-Queries

Future Work

- Enrich the Naive Bayes re-partitioning strategy for kNN stream join (theoretical and experimental analysis and proof – Accuracy, Load Balance)
- Enrich the query engine by providing operations such as FILTER, OPTIONAL, etc.

Future Applications

- Real Time Recommendation System Based on kNN
- Real Time Natural Language Processing System Based on RDF

Thank You!