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

Parallel and Continuous Join Processing for Data Streams

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-Liên 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

- 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)

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

Outline

Introduction

Definition: kNN

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

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

Introduction: Basic Idea

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

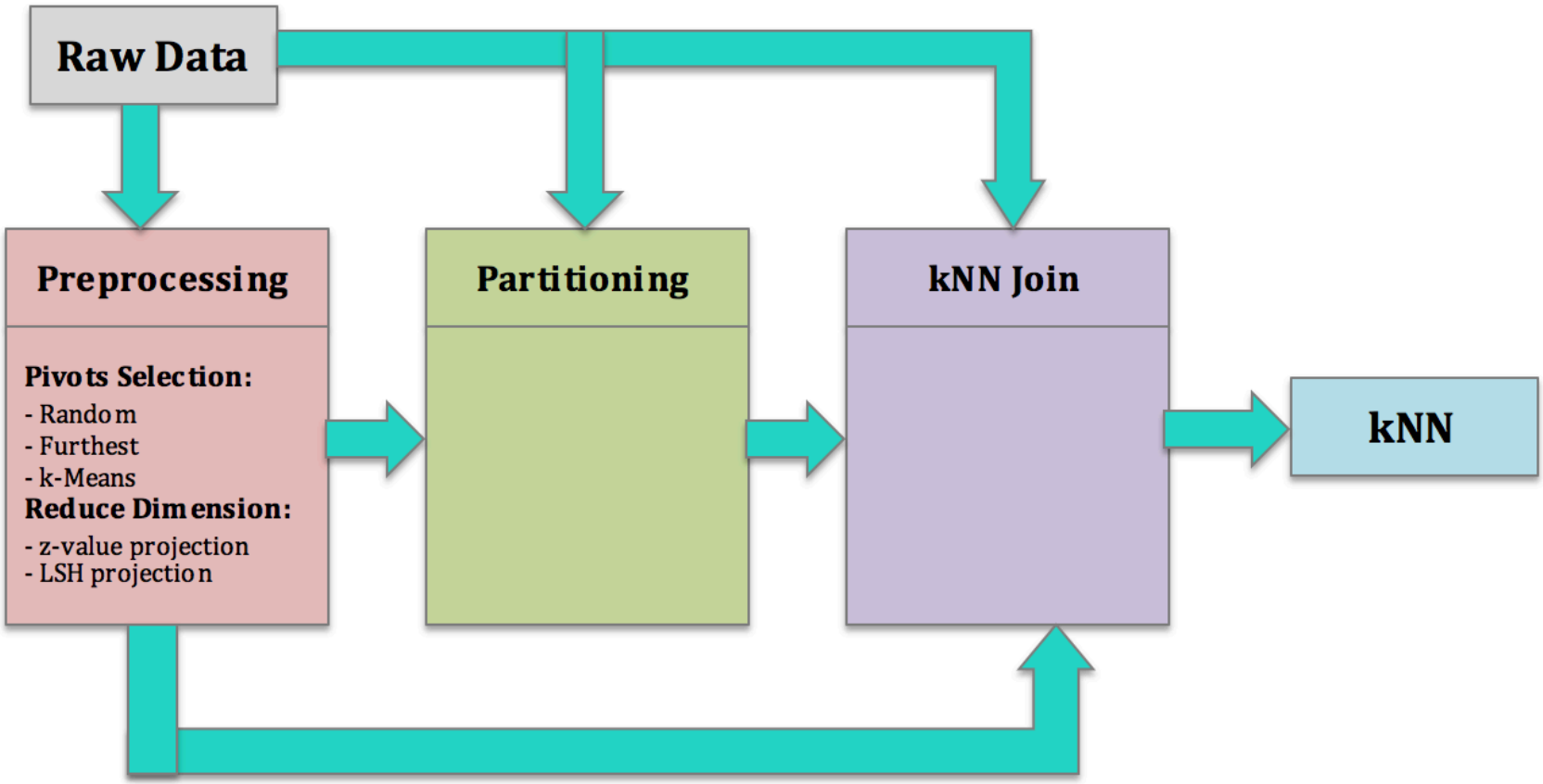
```
for(int r : R){
    for(int s : S){
        Distance(r, s);
    }
}
```

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

Outline

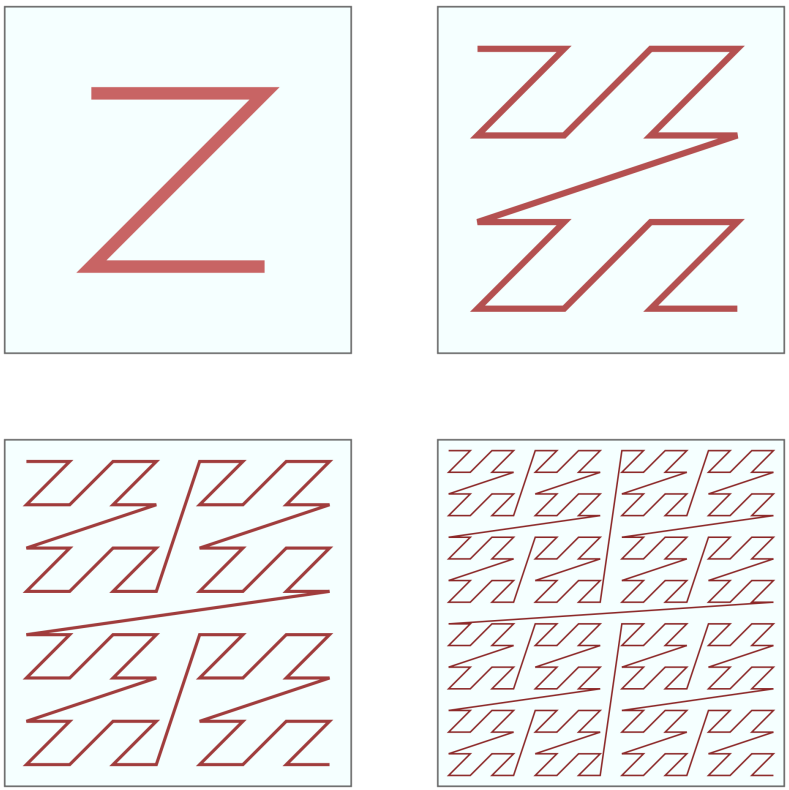
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

Parallel Workflow – Survey



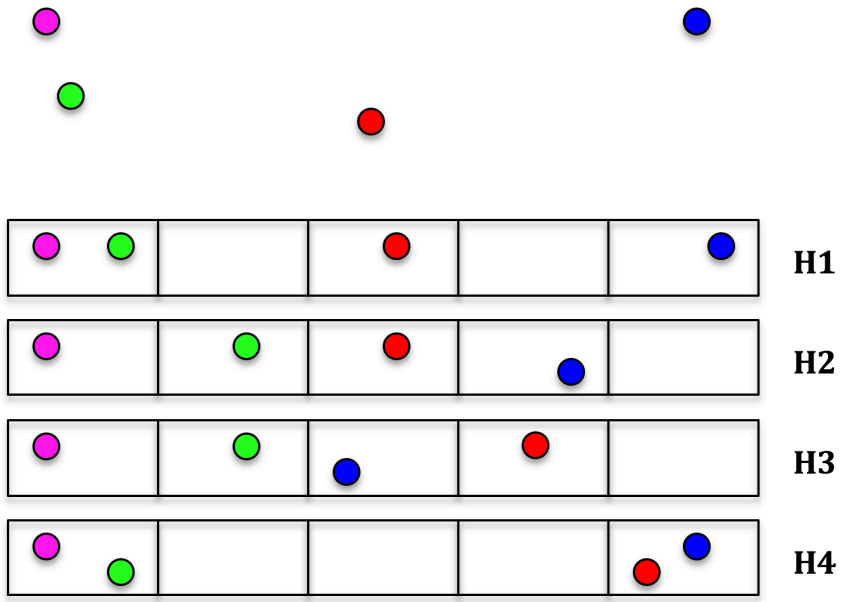
Data Preprocessing — Reducing the dimension of data

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



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

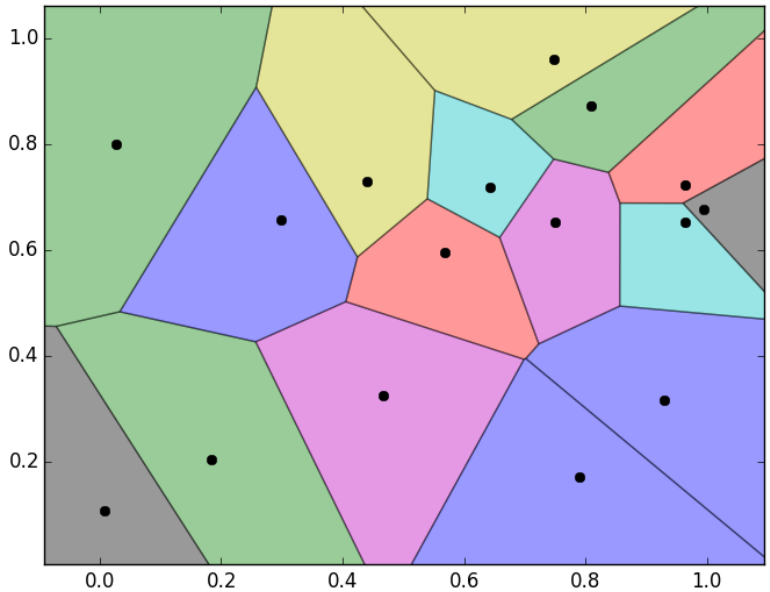


[LSH]: RankReduce - processing K-Nearest Neighbor queries on top of MapReduce, LSDS-IR 2010, Aleksandar Stupar et. al.

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

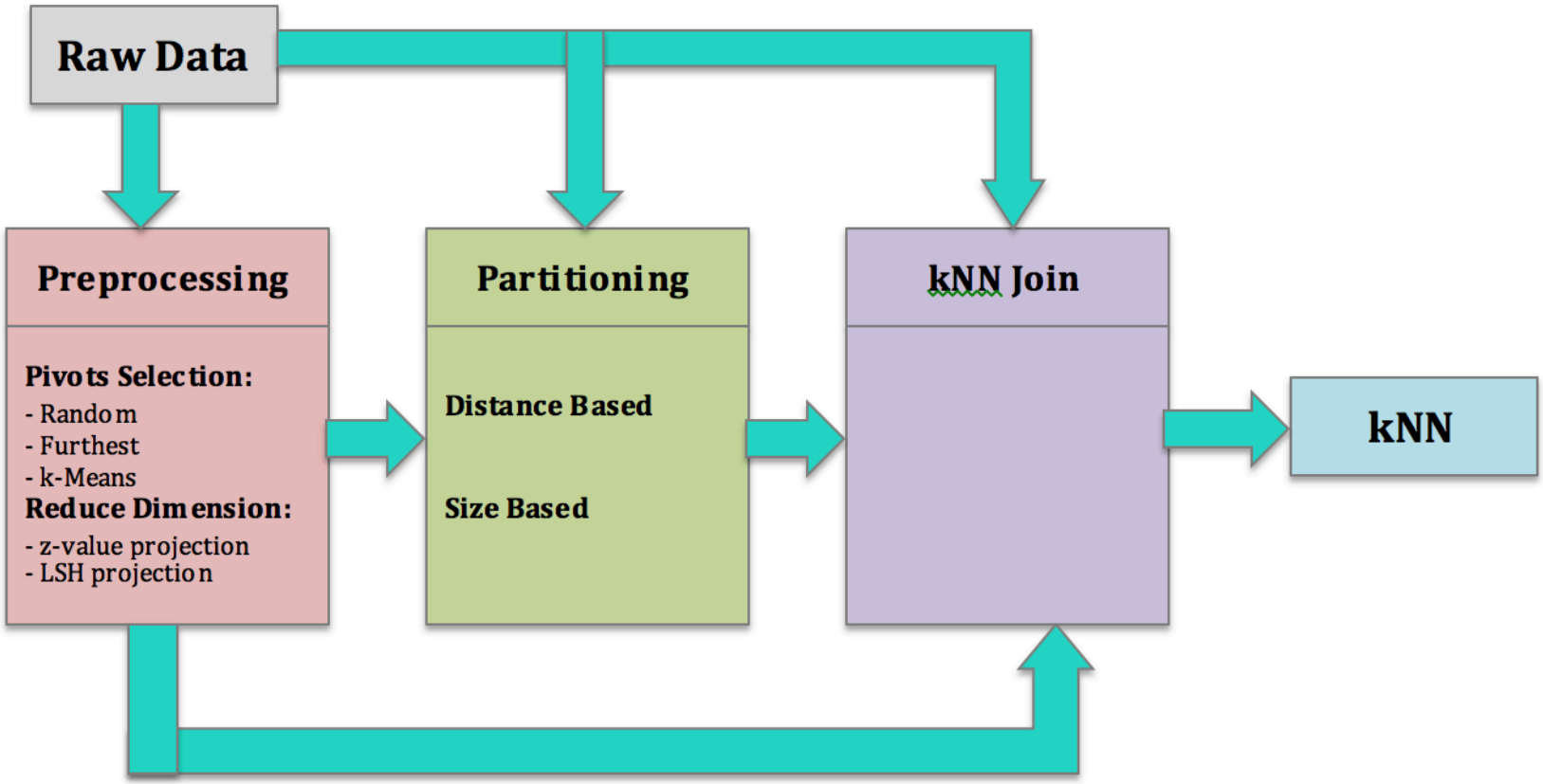
Voronoi Diagrams:

- Random Selection
- Furthest Selection
- K-Means Selection

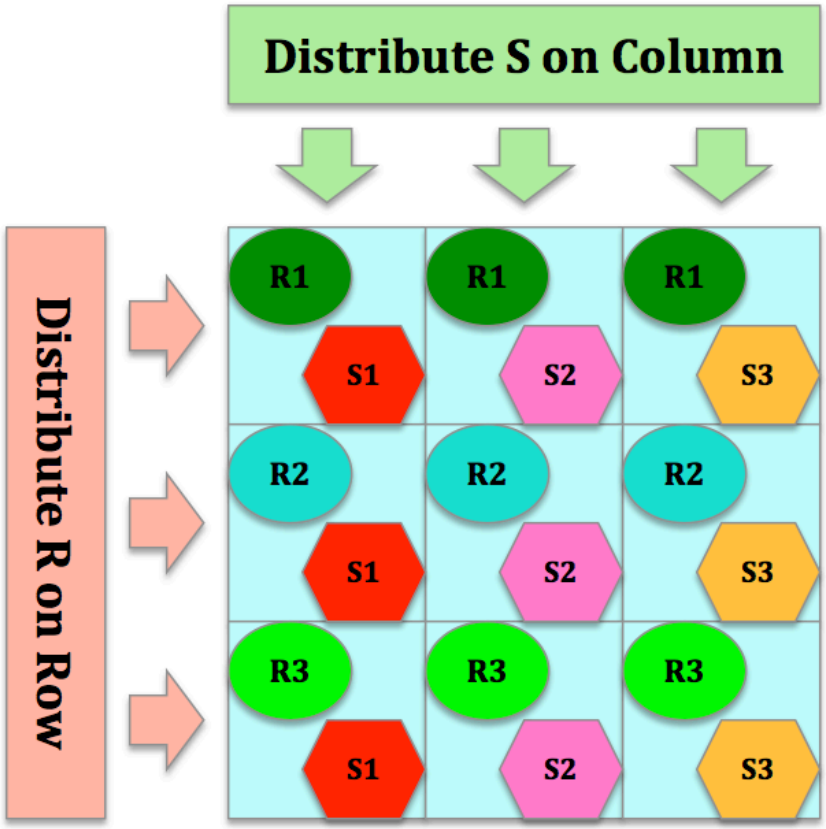


[voronoi]: Efficient Processing of k Nearest Neighbor Joins using MapReduce, VLDB 2012, Wei Lu et. al.

Parallel Workflow — Survey



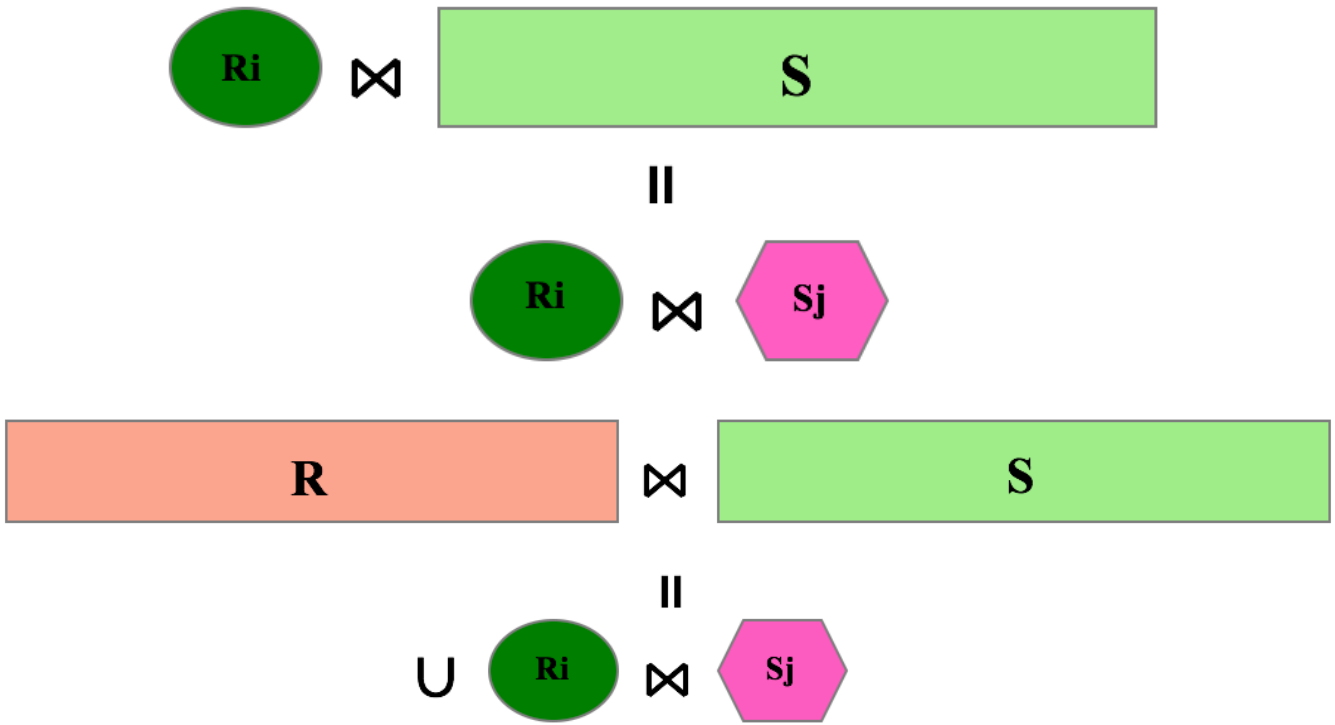
Data Partitioning — Basic Idea (Block Nested Loop)



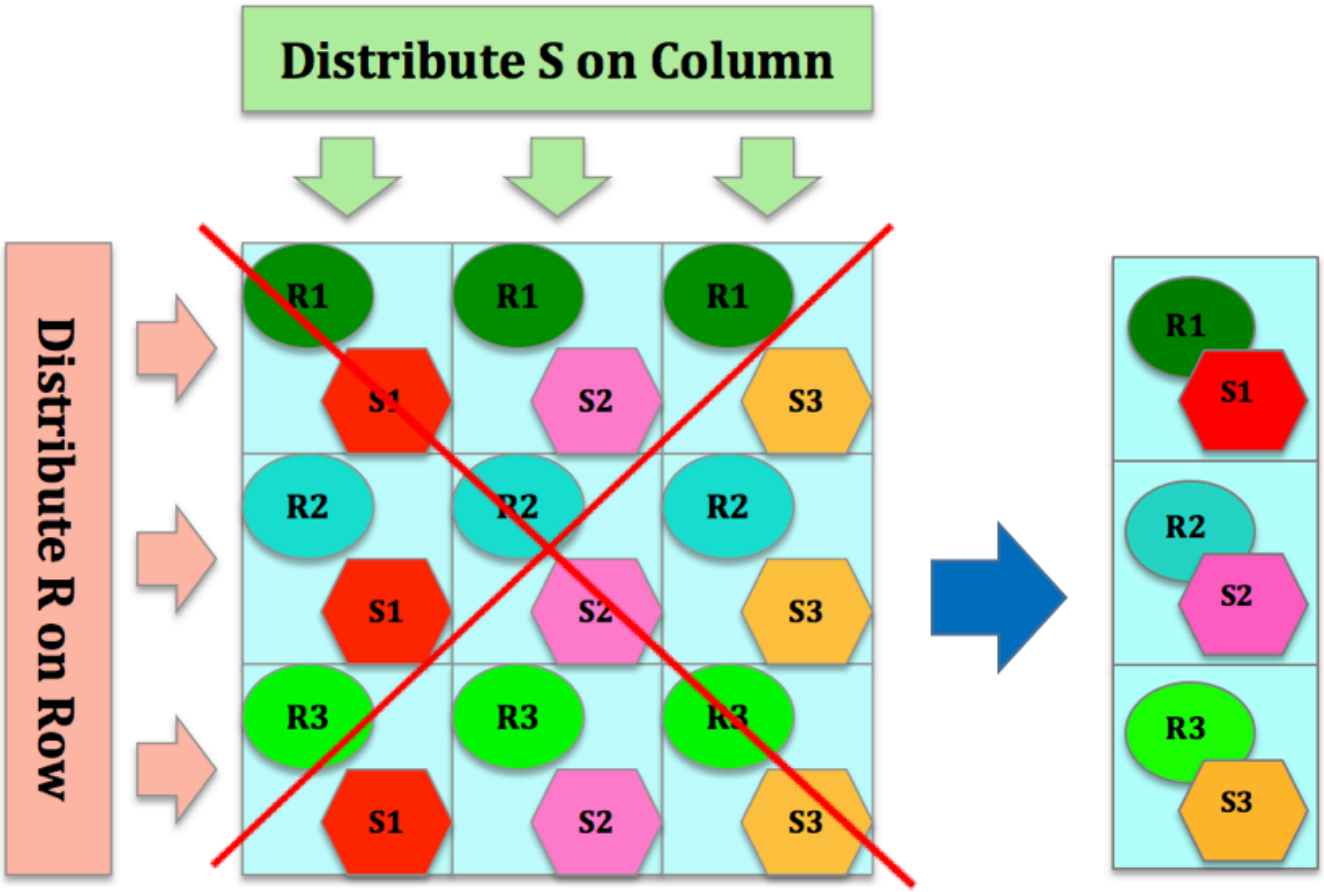
Problem: n^2 tasks for calculating pairwise distances; wastes a lot of hardware resources, and ultimately leads to low efficiency.

Data Partitioning – Motivation

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



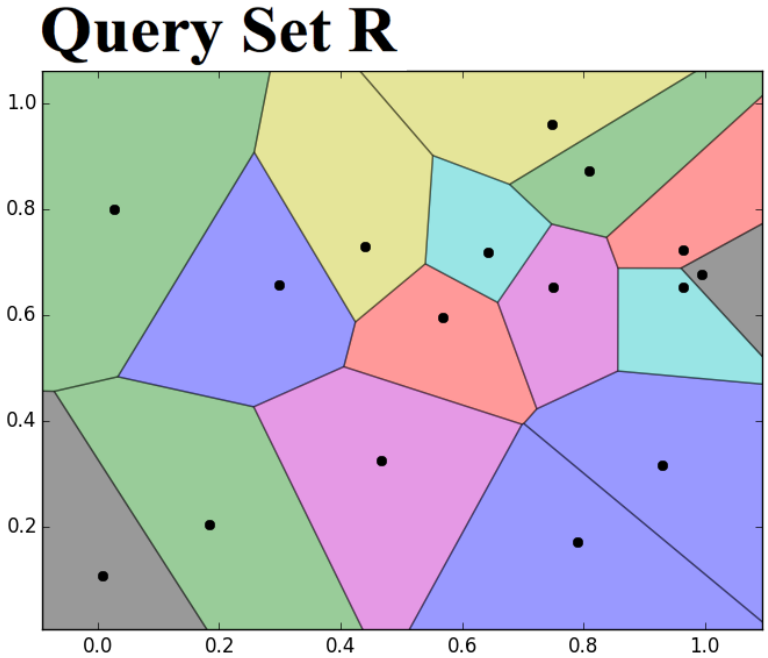
Data Partitioning – Motivation



Data Partitioning — Distance Based Partitioning Strategy — Voronoi Diagrams

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

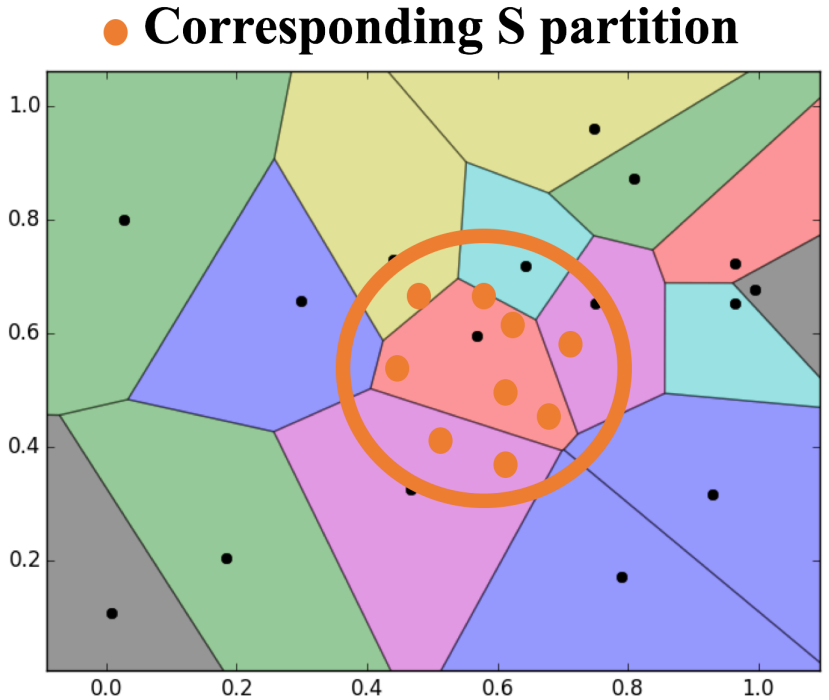


Take neighbor cells

Data Partitioning – Distance Based Partitioning Strategy – Voronoi Diagrams

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

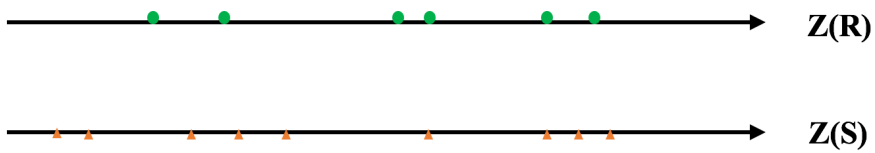


Take neighbor cells

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

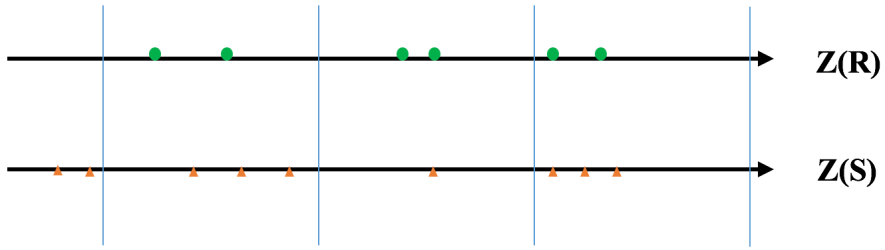


Take 3 partitions

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

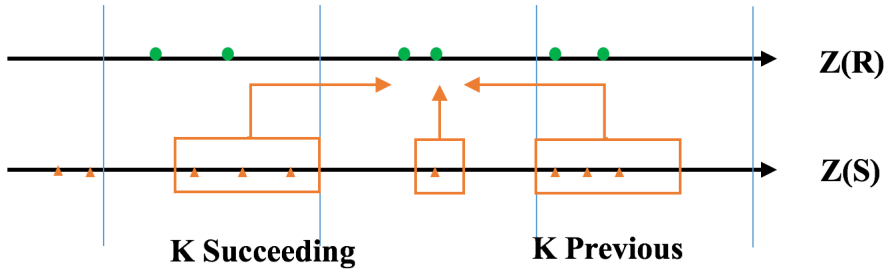


Take 3 partitions

Data Partitioning — Size Based Partitioning Strategy — Z-Value

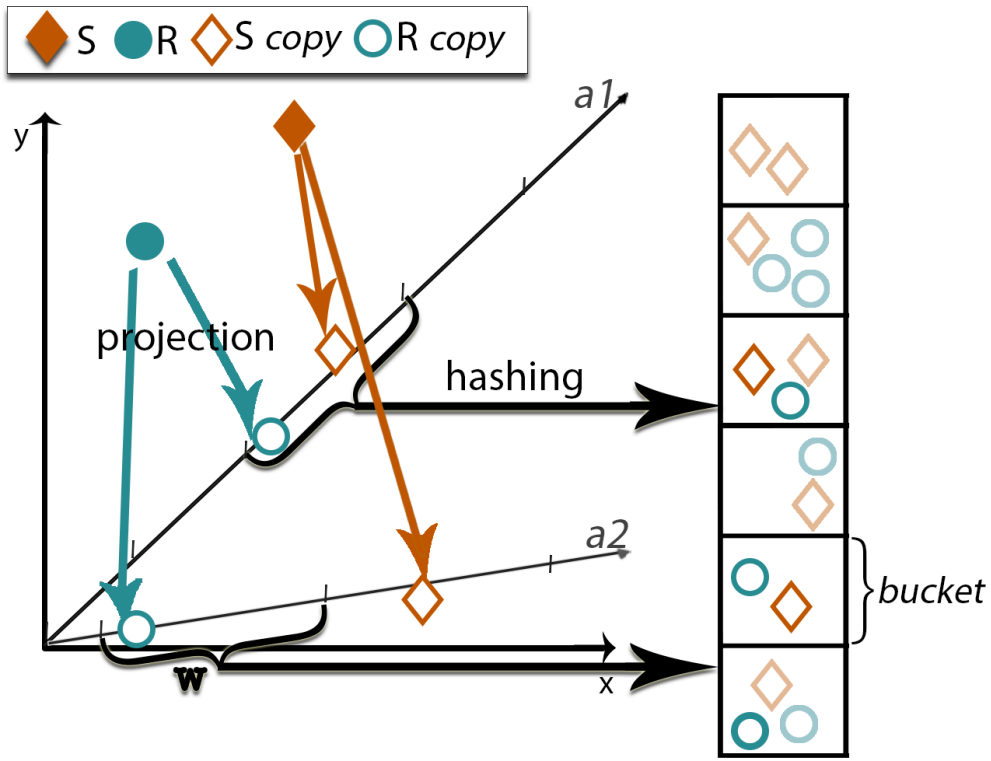
This strategy wants to make each partition have equal size in order to achieve a good load balance.

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



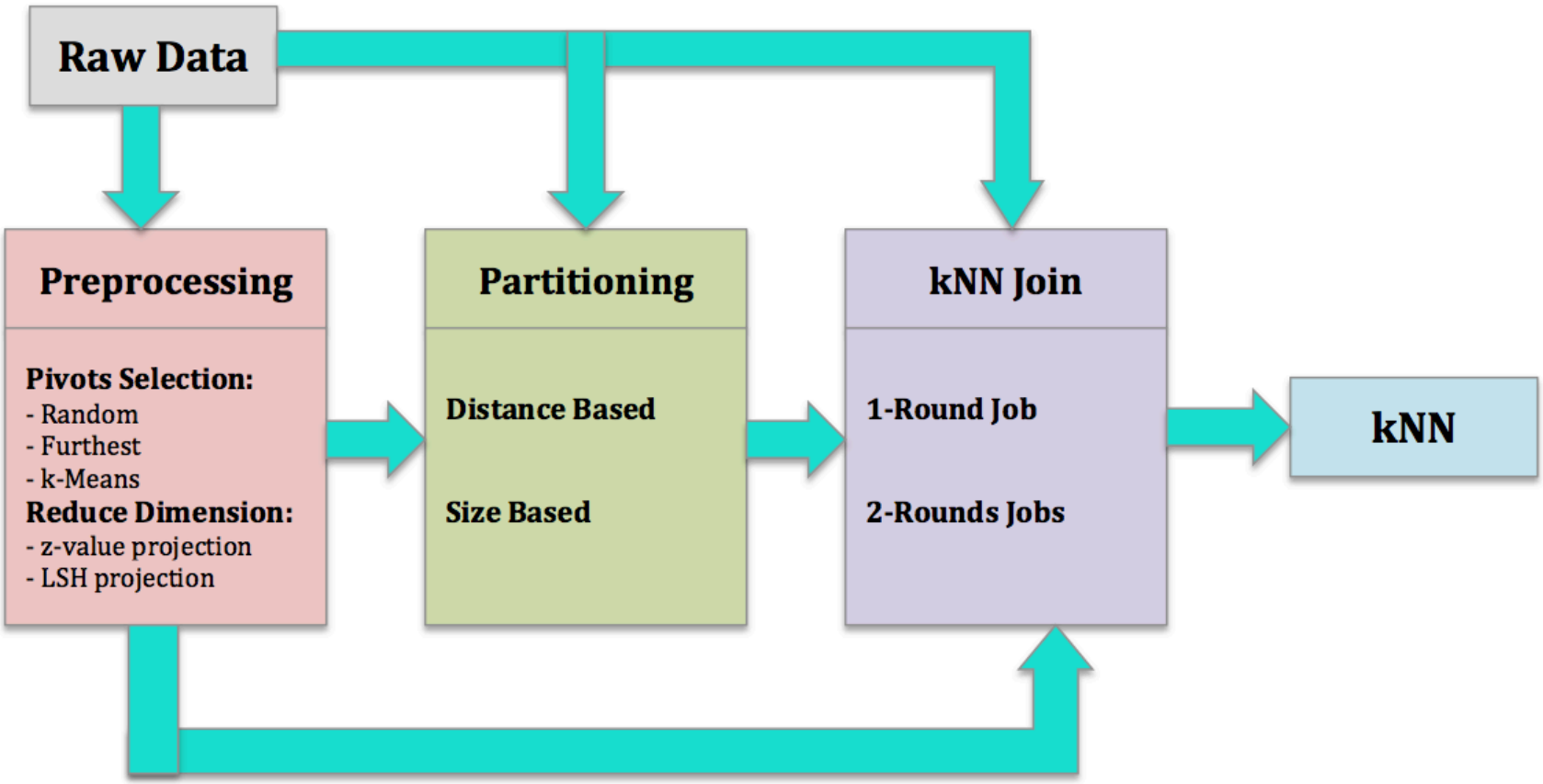
Take 3 partitions

Data Partitioning – Size Based Partitioning Strategy – LSH



Take neighbor buckets

Parallel Workflow – Survey



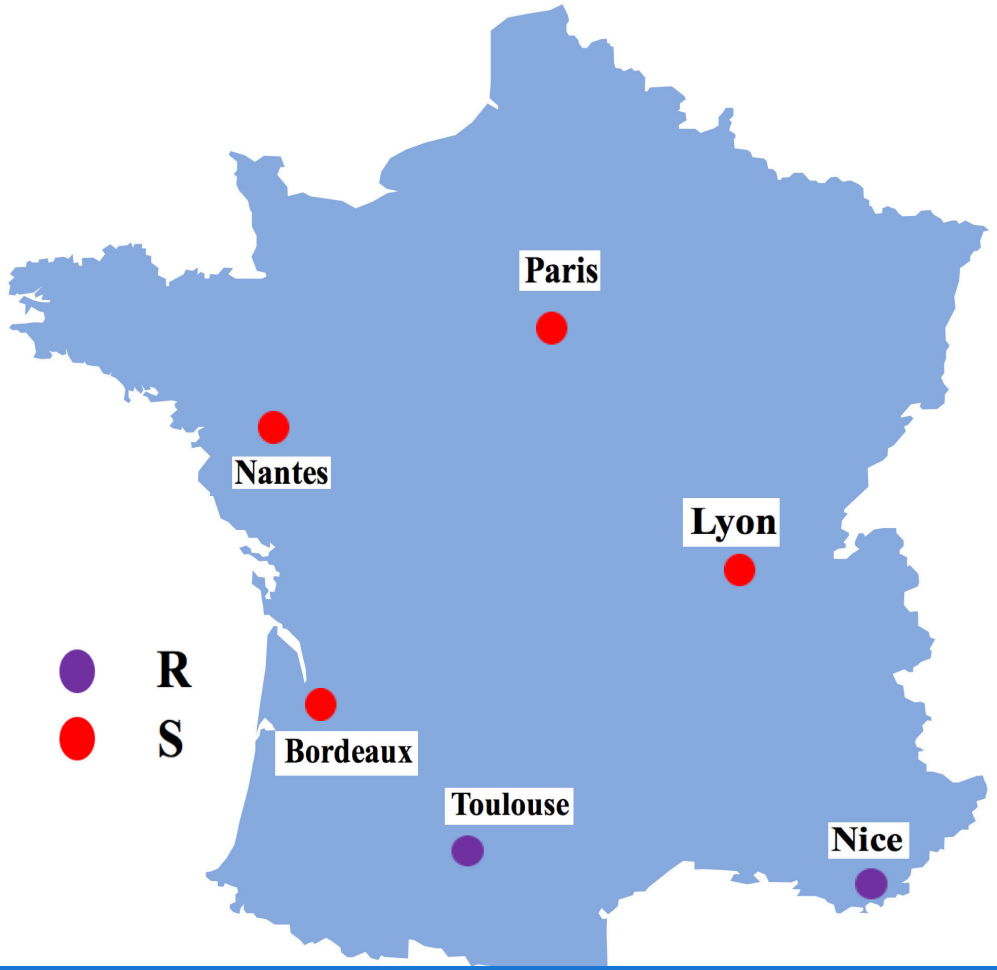
Computation

- 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.

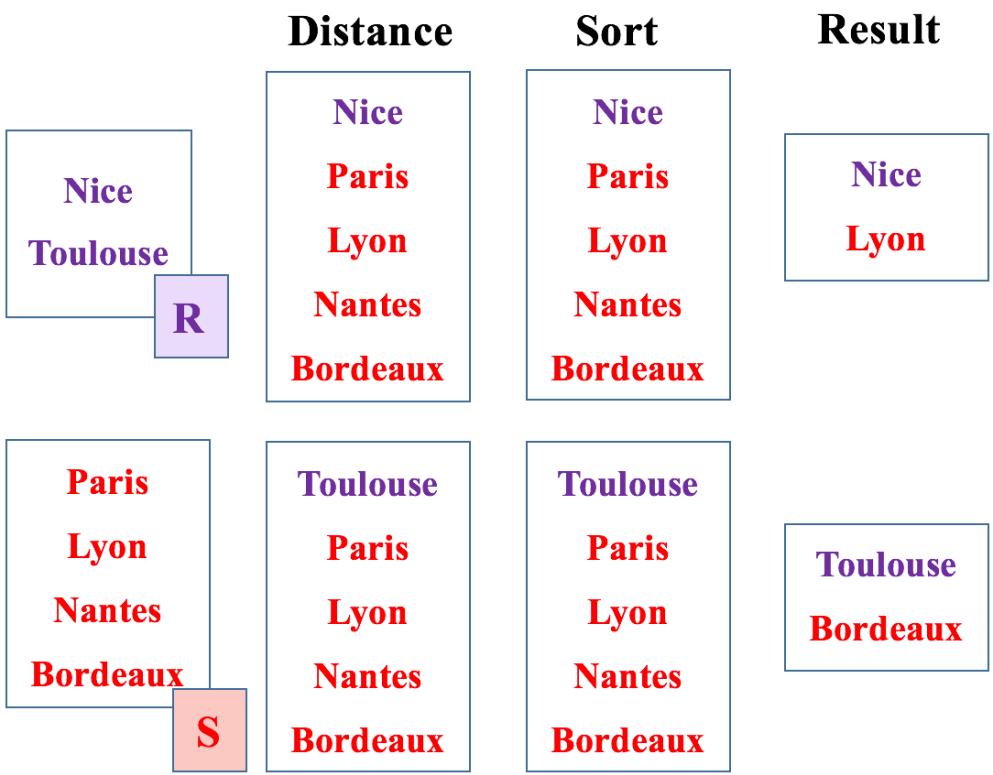
Computation – Example

For each city in R, find the nearest city in S. (1-NN)



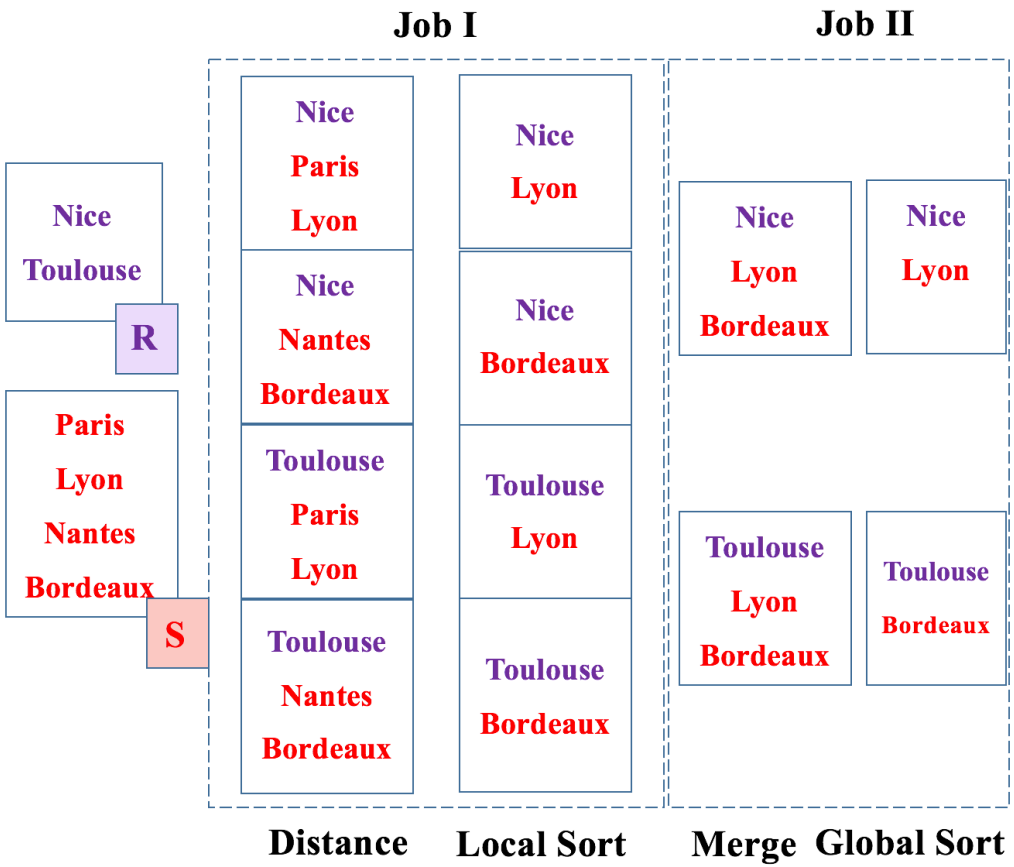
Computation – Example

One Job:



Computation – Example

Two Jobs:



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

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

Load Balance

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

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

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

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

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

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

Accuracy

Accuracy: the number of correct results we get in the end.

- **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 kNN$(\mathrm{R_i} \ltimes \mathrm{S})$:** the larger this number is, the more information is exchanged through the network.
- **The number of final candidates for each object $\mathrm{r_i}$:** 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

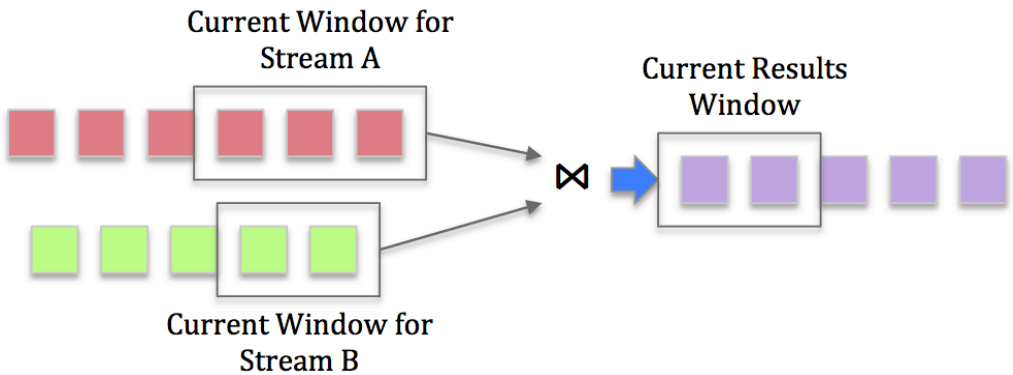
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

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



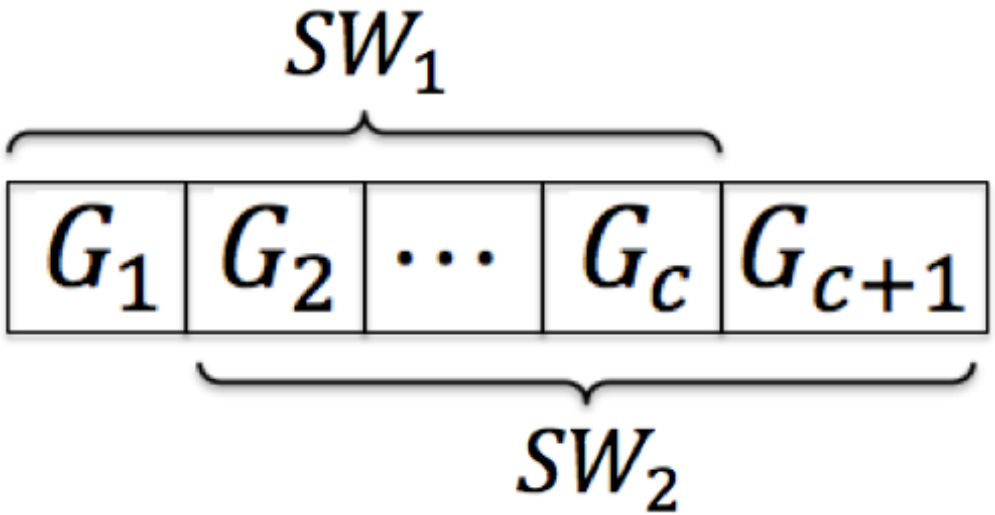
Sliding Window — Two Strategies

- **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
  - Find restaurant for moving users
- 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			
		S_1	S_2	...	S_n
R in i^{th} Generation	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 re-partition 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 (3 replications, 1 slot per node)

Datasets

- **OpenStreetMap** Geo dataset contains geographic XML data in two dimensions — Low Dimension
- **Caltech 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

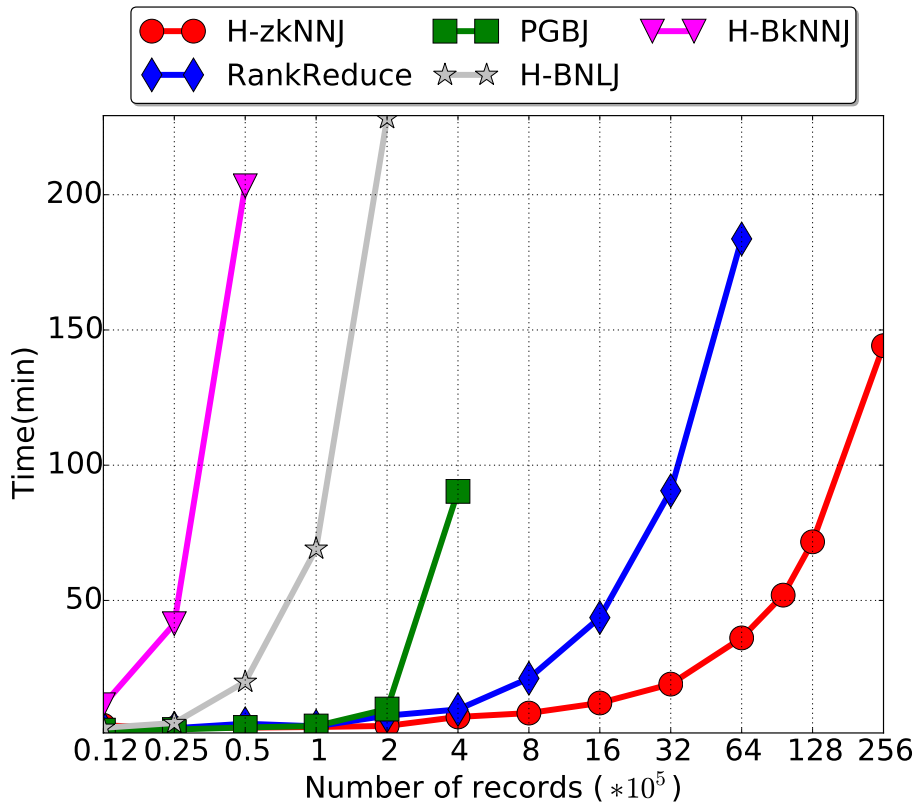
Methods Evaluated

- **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

Evaluation Result — Verify the theoretical Analysis

Execution Time for Geo dataset (2 dimensions):

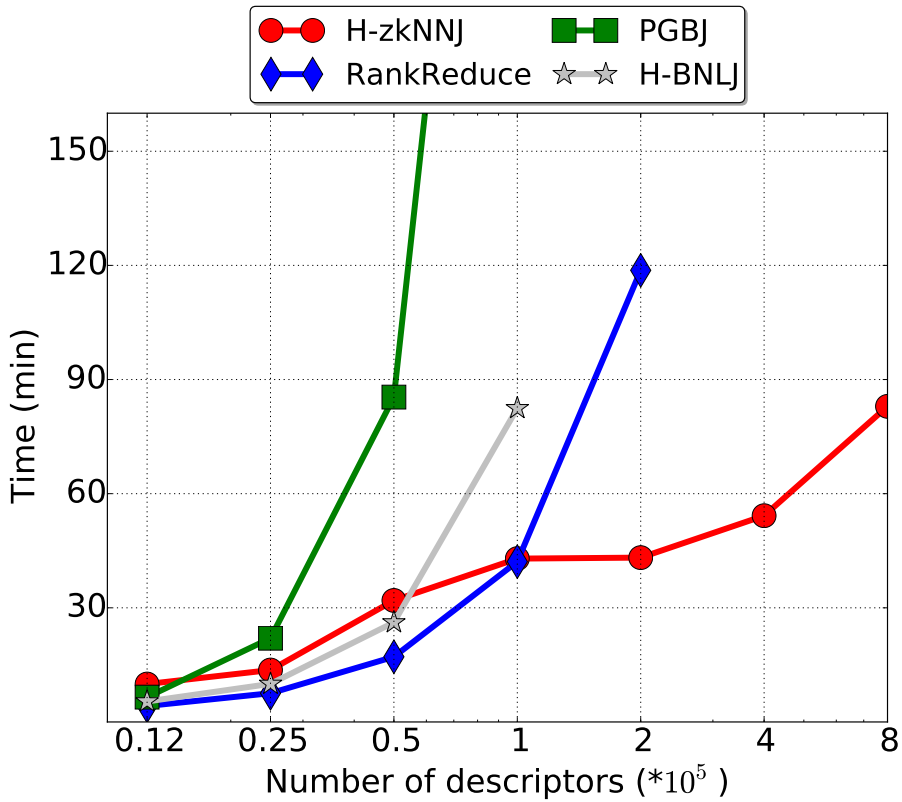
- H-BkNNJ: Naive
- H-BNLJ: Block Nested Loop
- PGBJ: Voronoi
- H-zkNNJ: z-value
- RankReduce: LSH



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



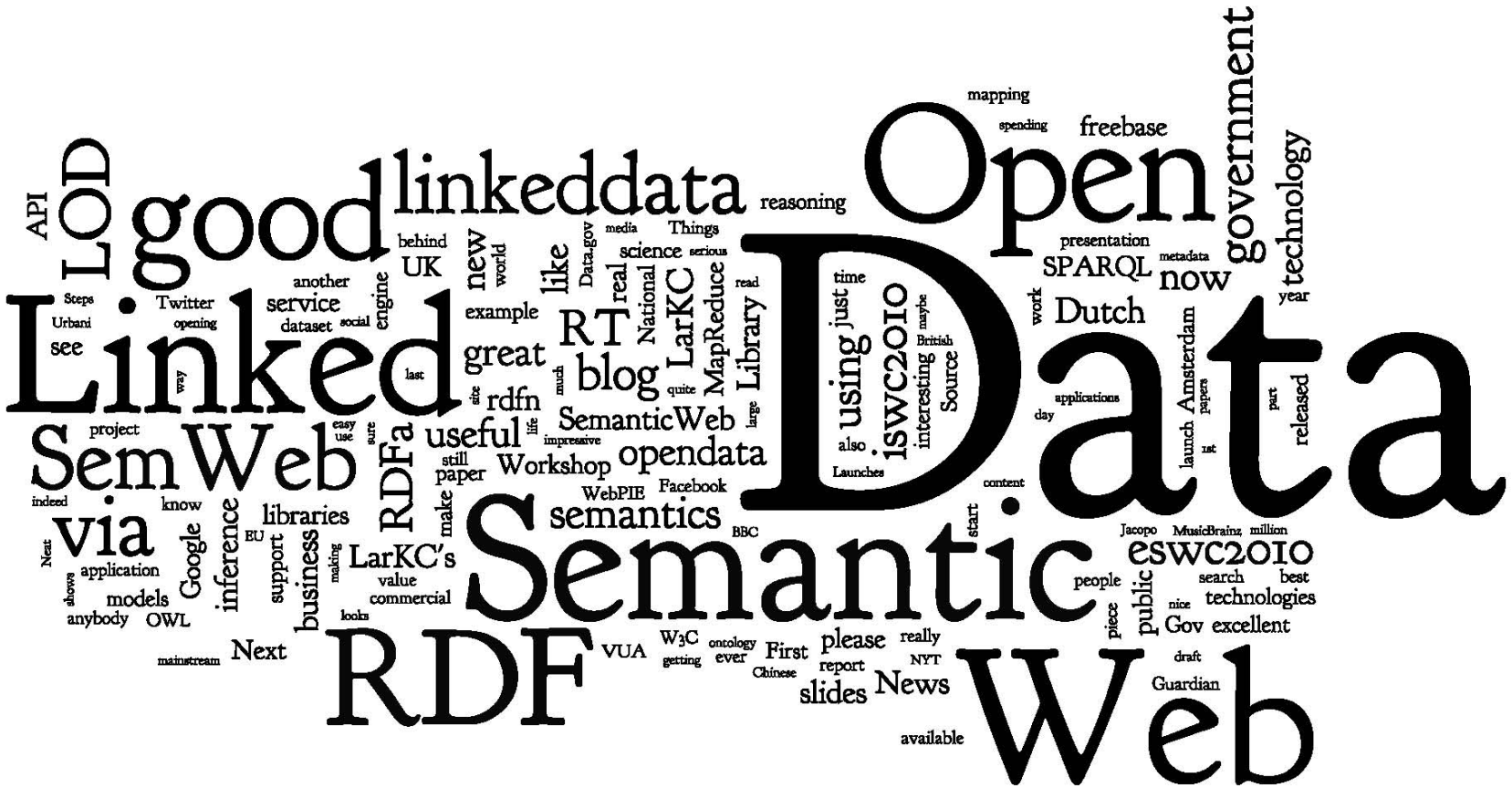
Outline

- 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

- **R**esource **D**escription **F**ramework is a data standard proposed by W3C
- 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>)

Introduction — SPARQL Query Language

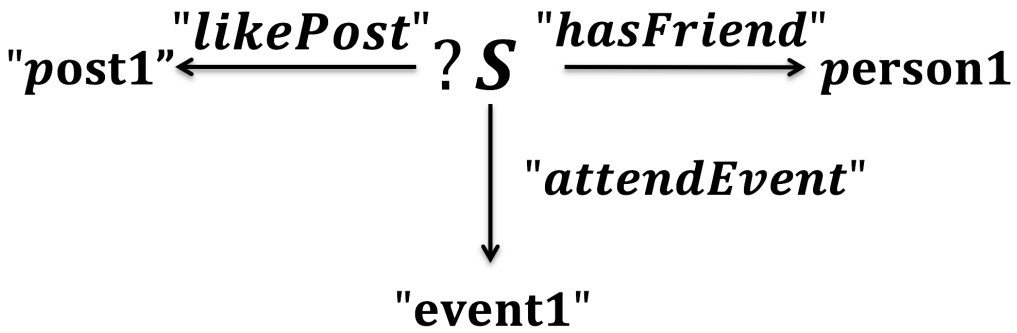
- SPARQL is a W3C recommendation query language for querying RDF data.
- The basic component of a SPARQL query is the triple patterns.
- A triple pattern is a special kind of triple where S, P and O can be either a literal or a variable.

An Example (Triple Pattern Representation):

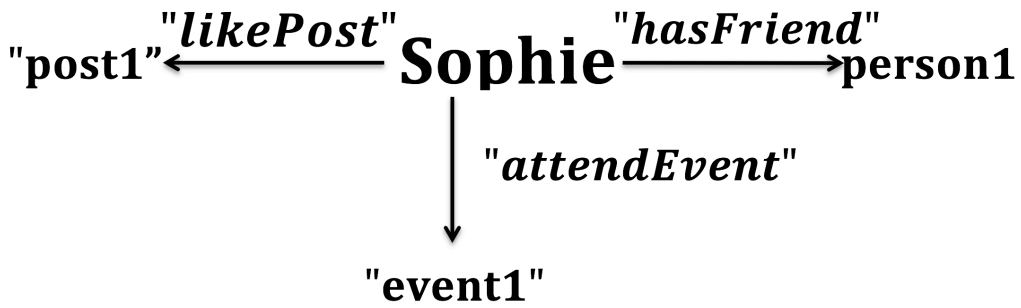
```
SELECT ?S
WHERE{
  Q1    ?S "hasFriend" person1 .
  Q2    ?S "likePost" "post1" .
  Q3    ?S "attendEvent" "event1"
}
```

Introduction — SPARQL Query Example

Graph Representation for SPARQL Query:



Graph Representation for RDF Data:

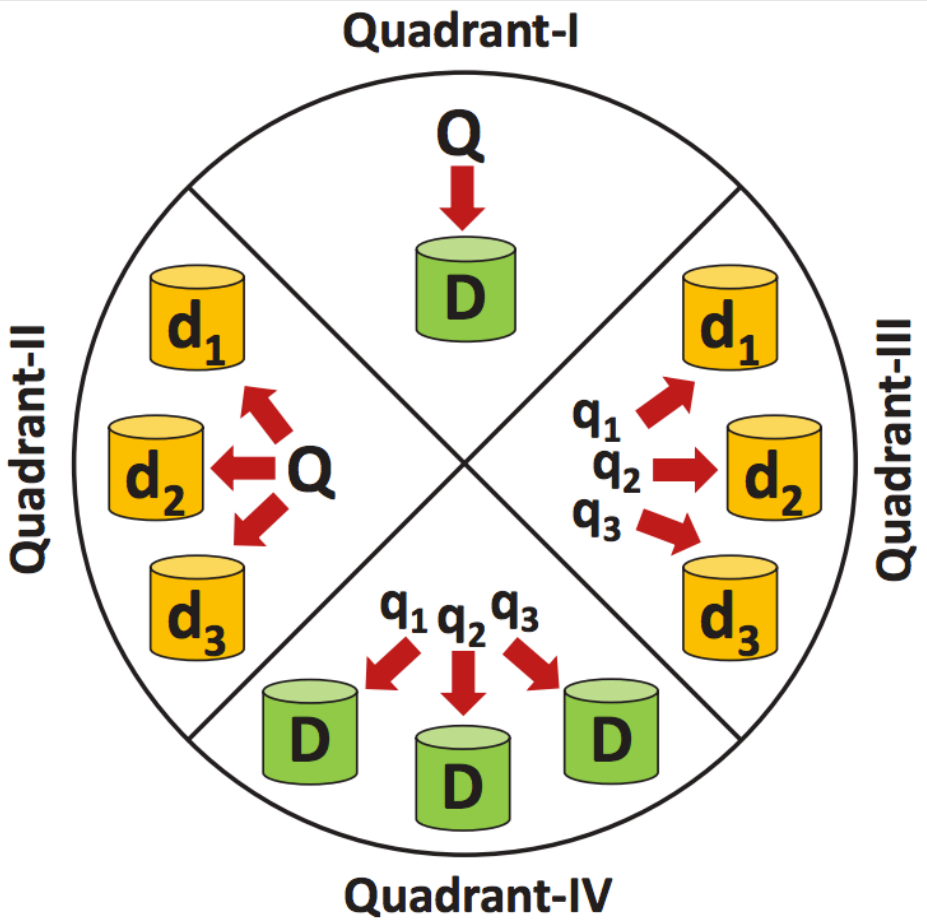


Query Driven Join

- Data format does not change
- Query changes

Related Works – 4 Types of Processing

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



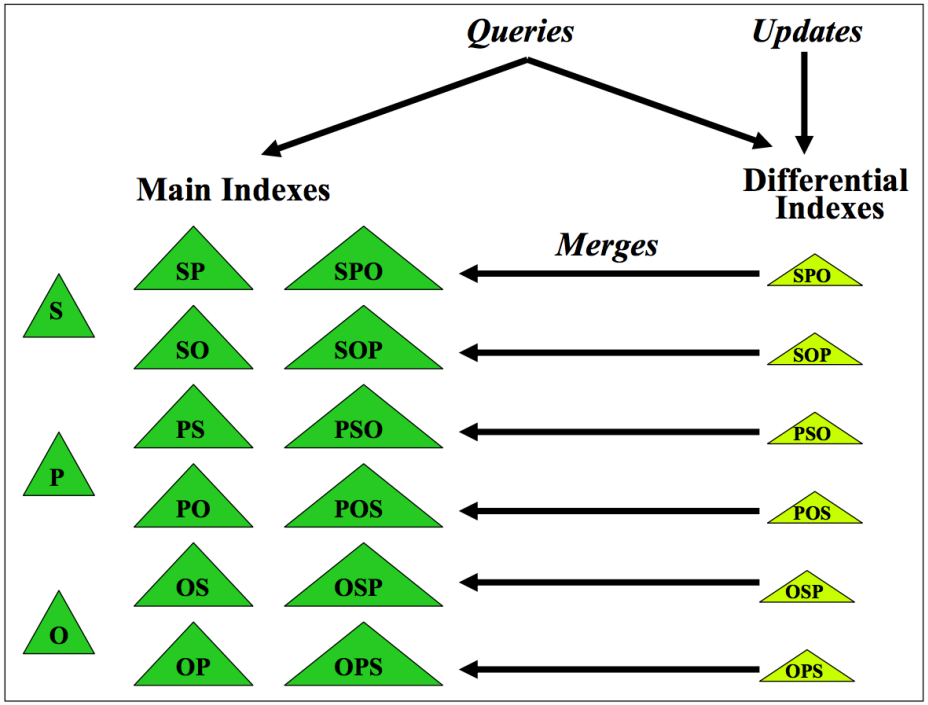
DREAM: distributed RDF engine with adaptive query planner and minimal communication, PVLDB 2015, Mohammad Hammoud et. al.

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
  - Too many 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

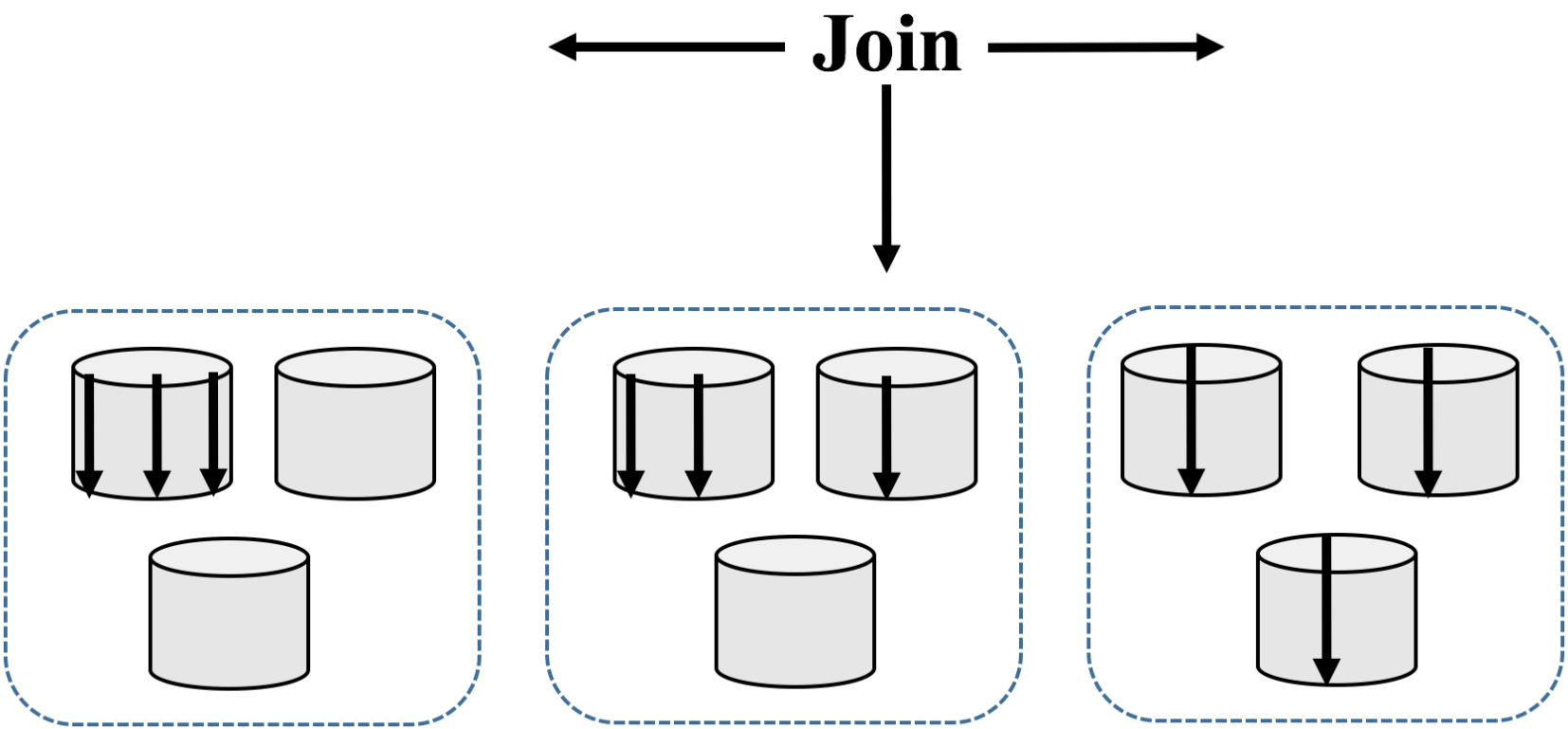
- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

Query Decomposition

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

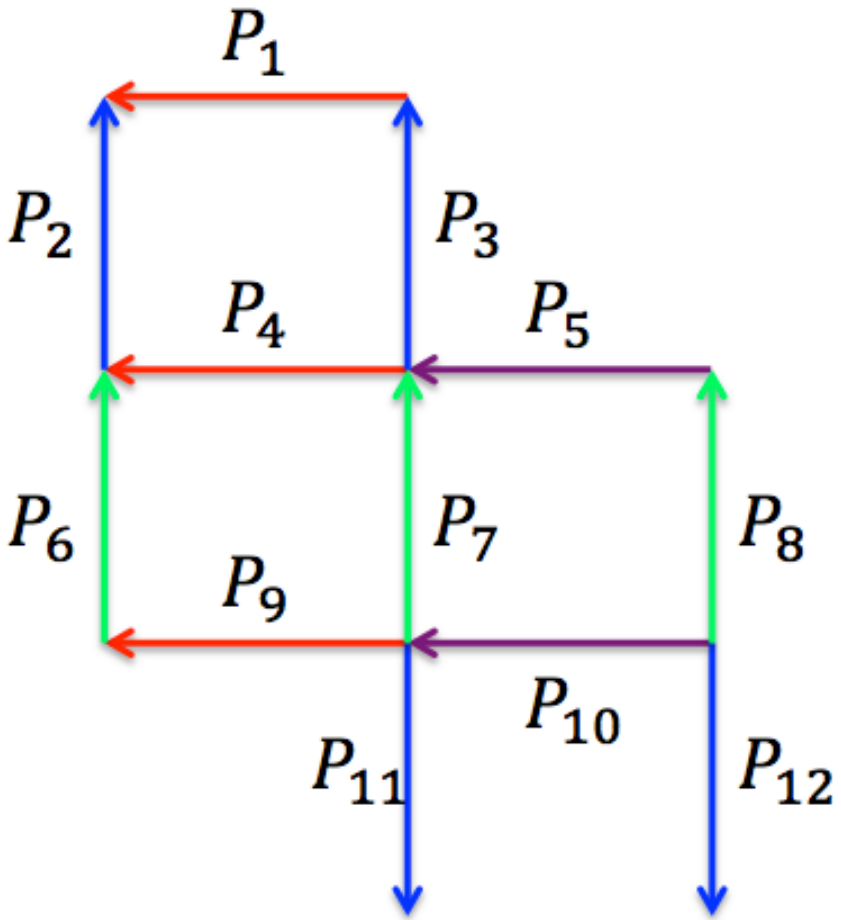
- It is simple, and does not require any complicated computations.
- The performance or accuracy of this method does not depend on the index or replication of data.
- The number of triple patterns is limited

Sub-Query Scheduling



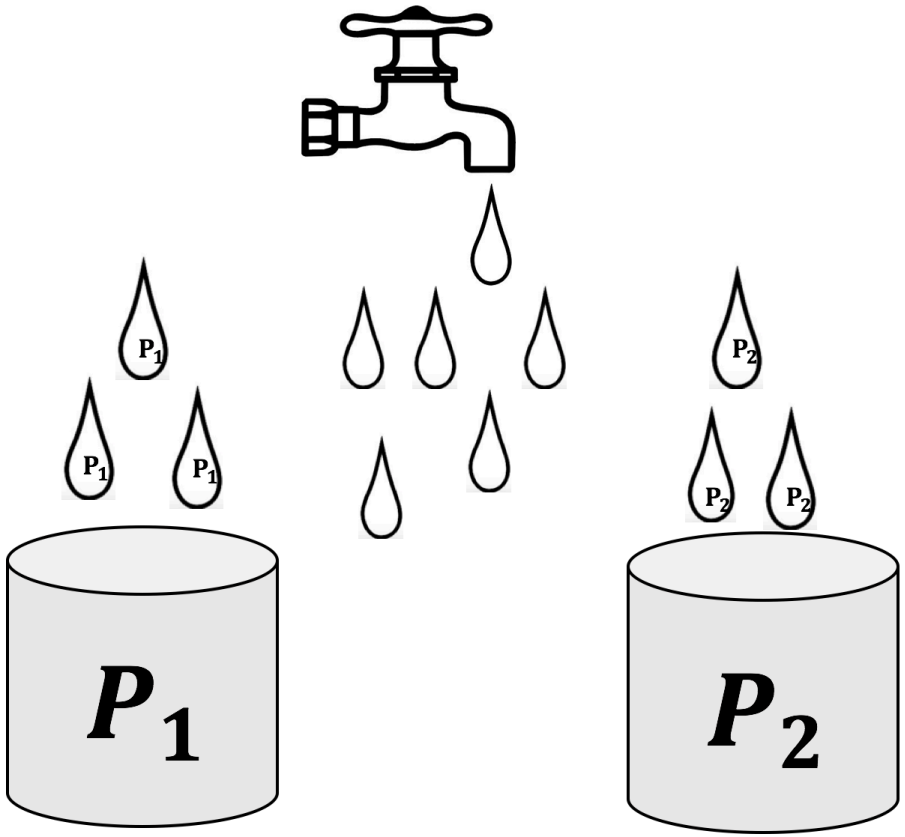
Sub-Query Scheduling

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.



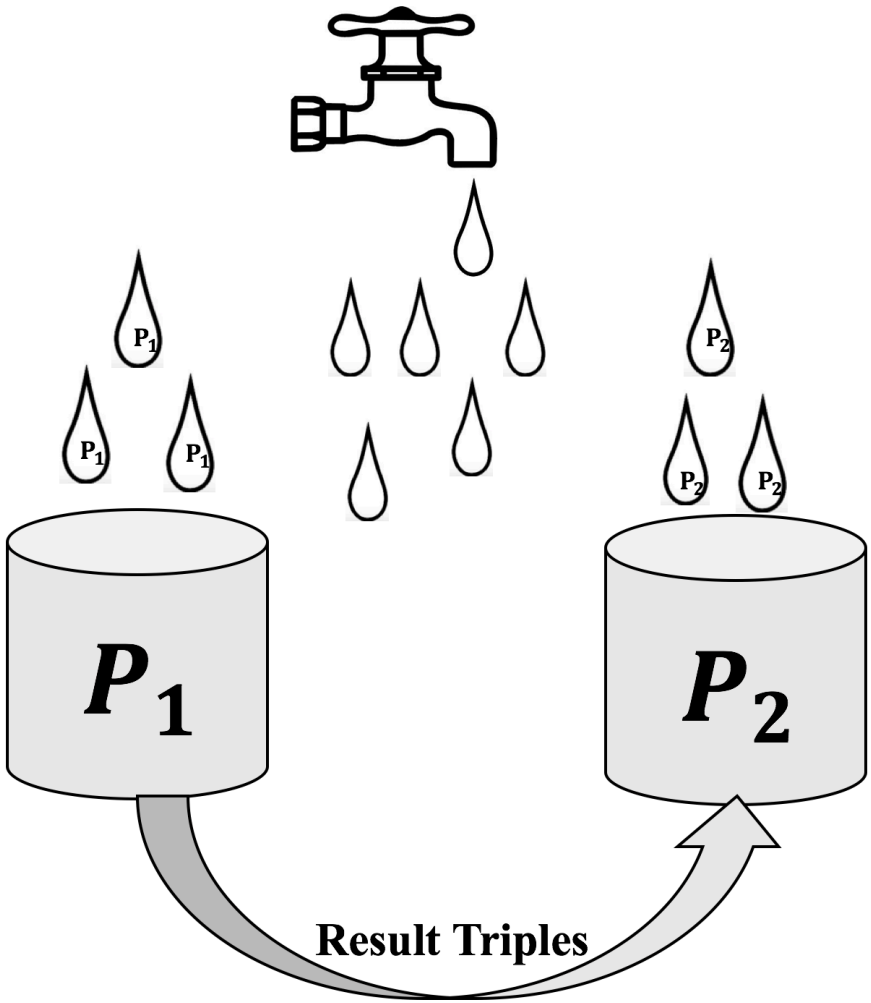
Outline

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

Challenges

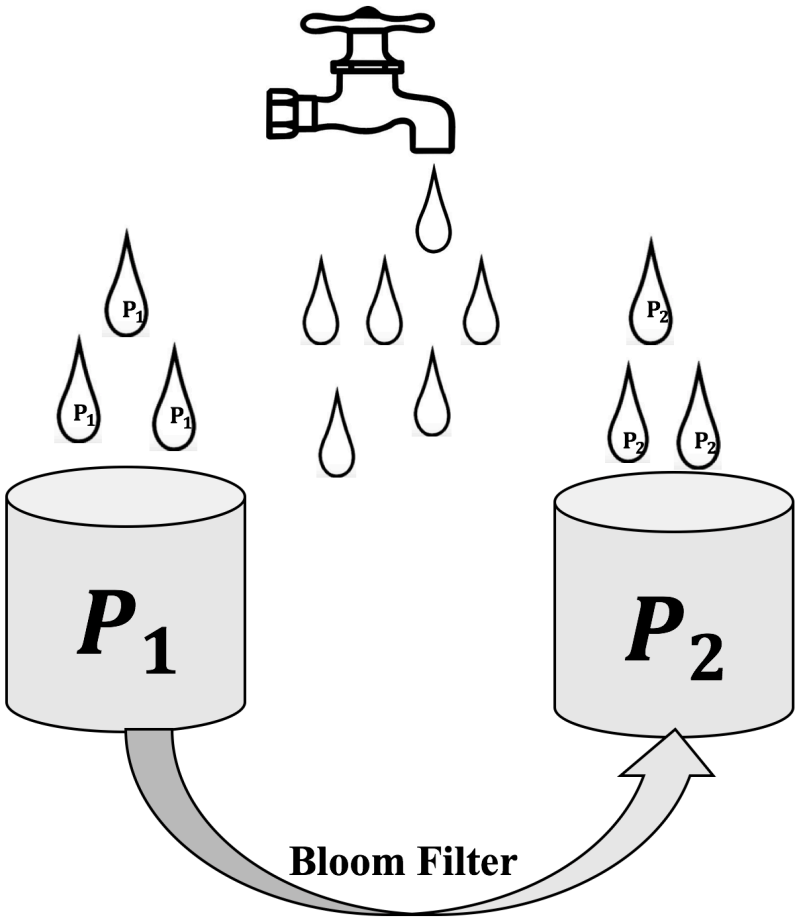
- Communication among nodes
- Join of the intermediate results produced by each triple pattern
- Order of sending and receiving information

Communication



Communication

Do not send triples, send Bloom Filters



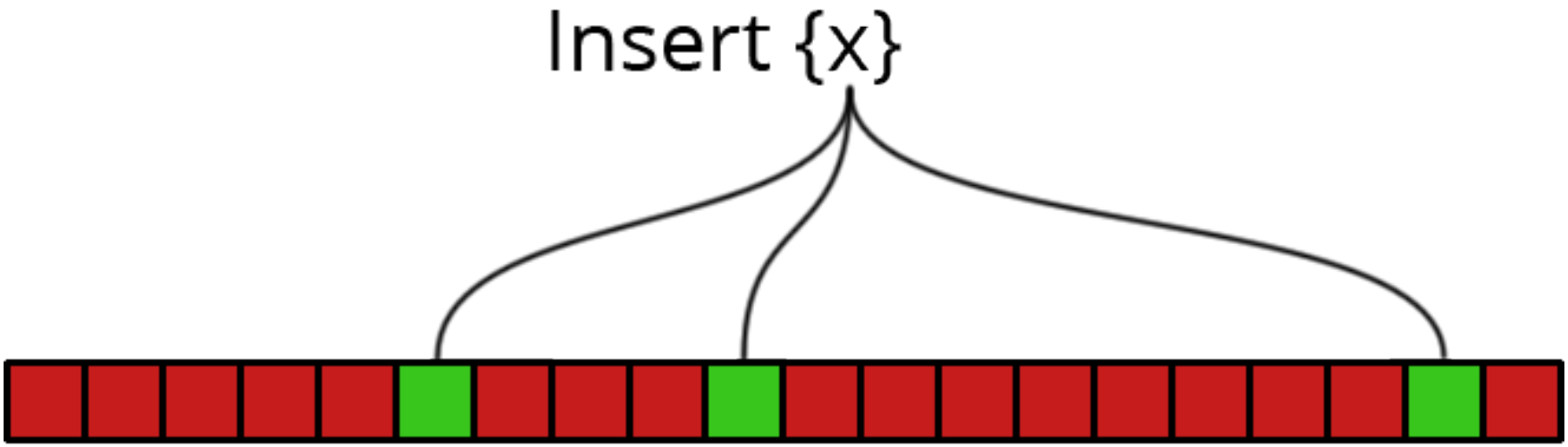
Communication among nodes – Bloom Filter

m = 20 bits

k = 3 hash functions

n = 3 insertions

Communication among nodes – Bloom Filter

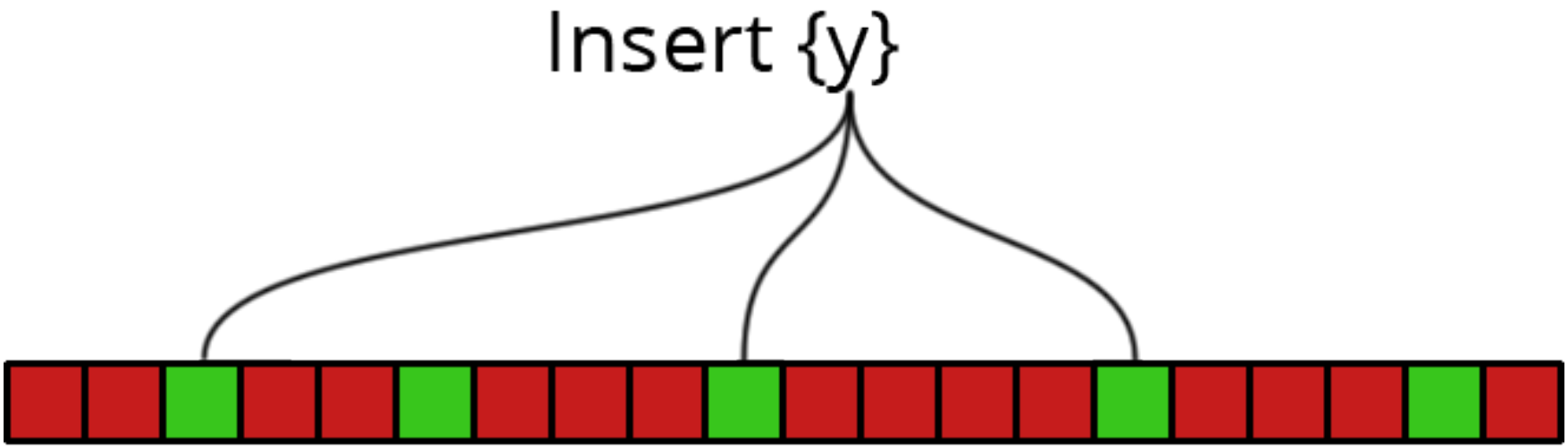


m = 20 bits

k = 3 hash functions

n = 3 insertions

Communication among nodes – Bloom Filter

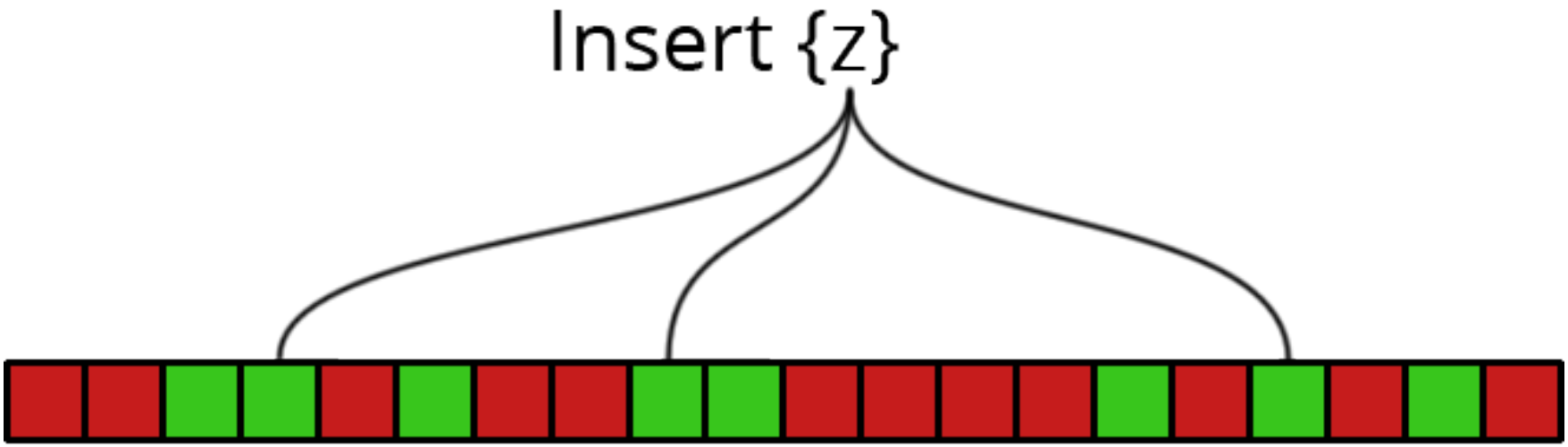


m = 20 bits

k = 3 hash functions

n = 3 insertions

Communication among nodes – Bloom Filter

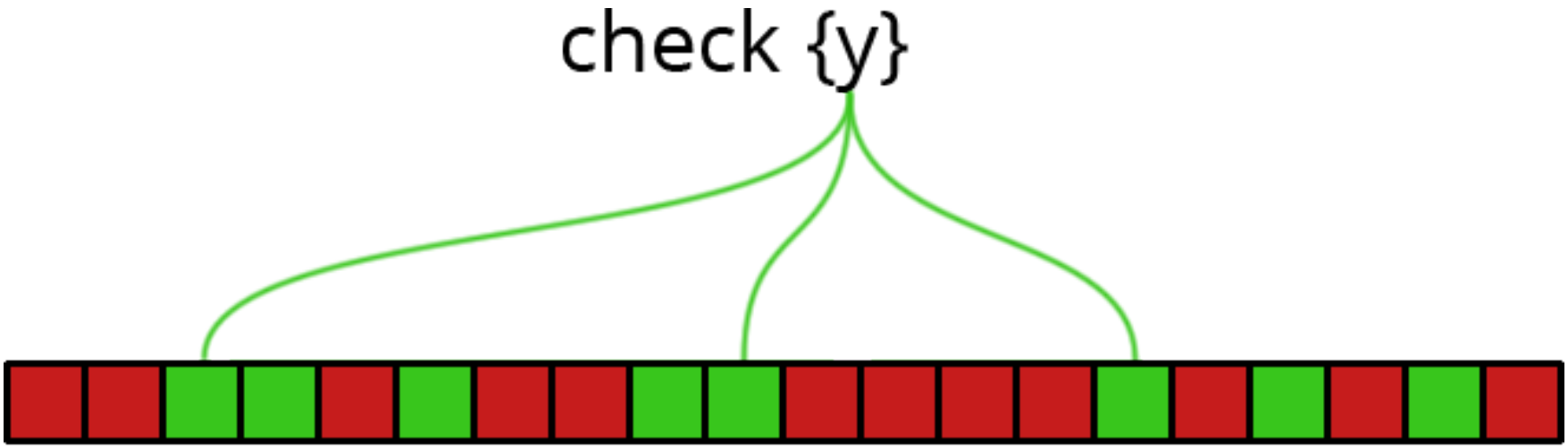


m = 20 bits

k = 3 hash functions

n = 3 insertions

Communication among nodes – Bloom Filter

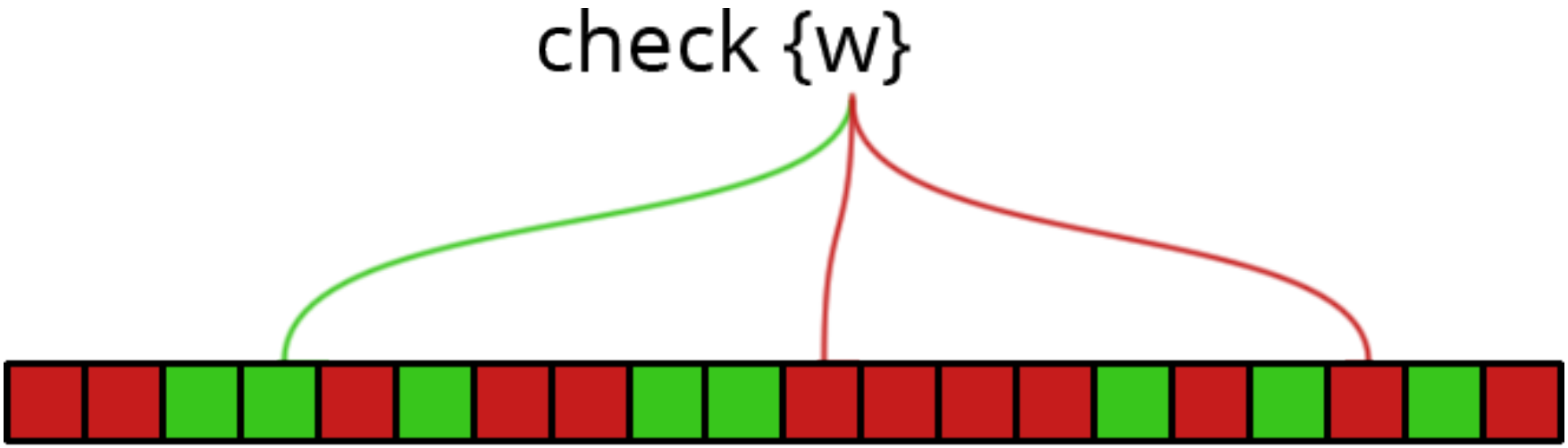


m = 20 bits

k = 3 hash functions

n = 3 insertions

Communication among nodes – Bloom Filter

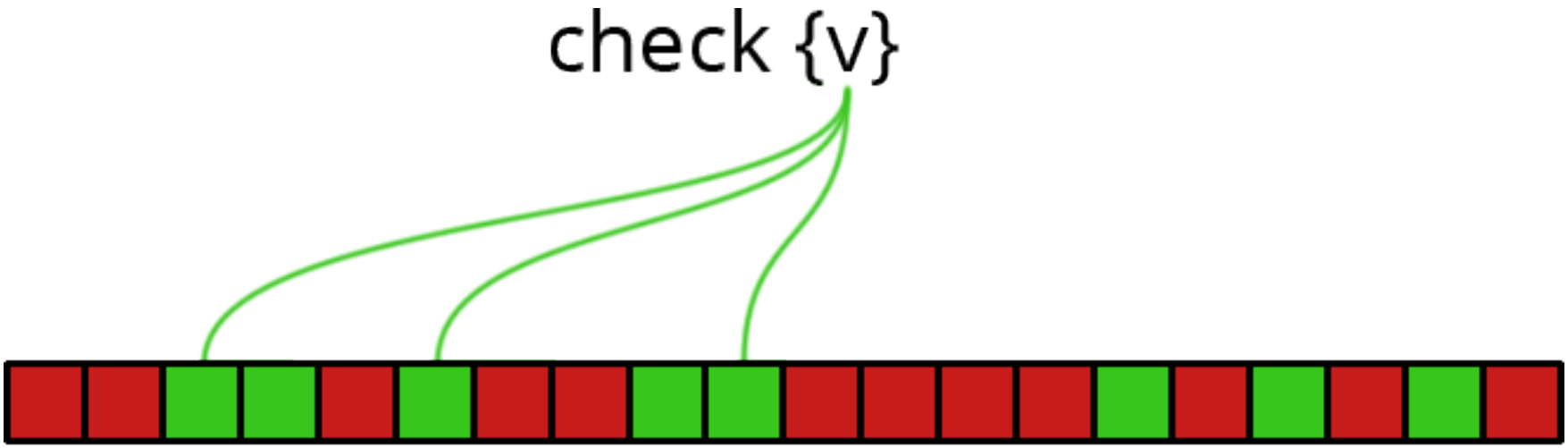


m = 20 bits

k = 3 hash functions

n = 3 insertions

Communication among nodes – Bloom Filter



m = 20 bits

k = 3 hash functions

n = 3 insertions

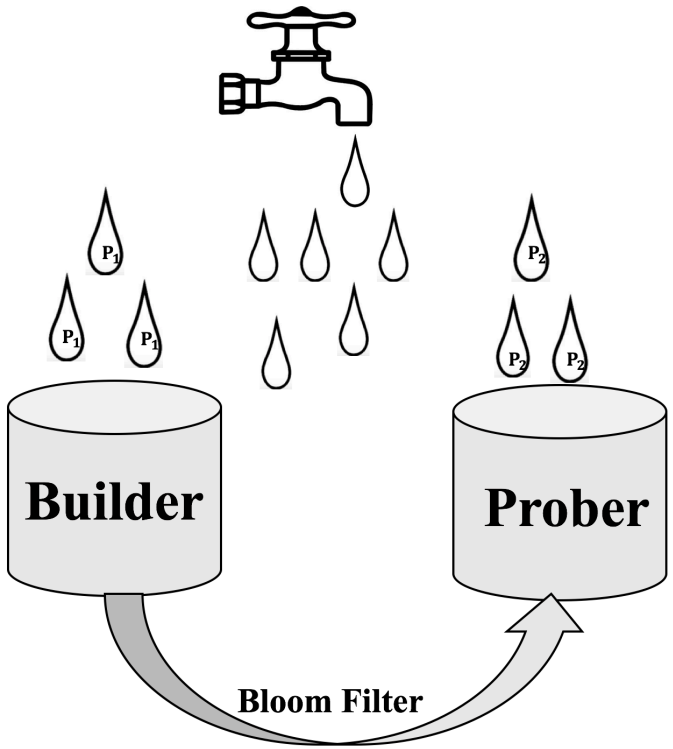
Communication among nodes – Bloom Filter

False Positive Rate

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

Bloom Filter – Build and Probe

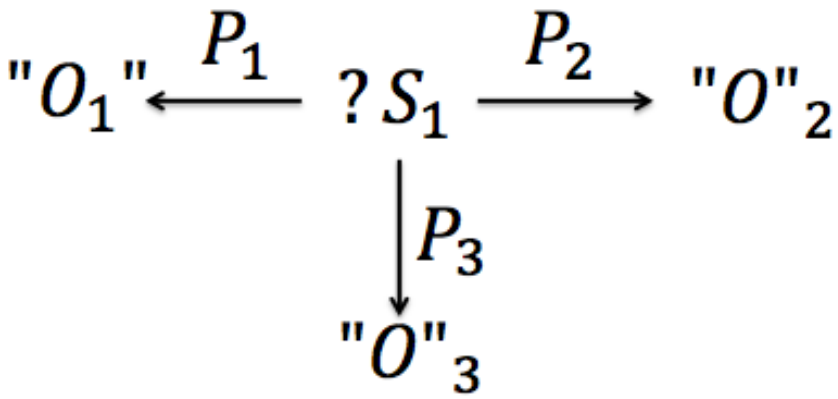
- **Builder:** inserts the results of the triple pattern to the Bloom Filter.
- **Prober:** checks the existence of the results of the triple pattern in the Bloom Filter

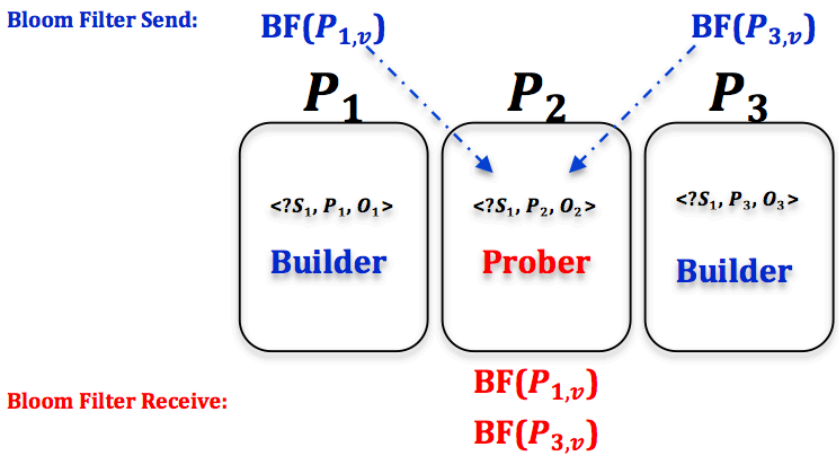


Question: Which triple patterns should be **Builders**, and which ones should be **Probers**?

The join of the intermediate results — Structure Based Rules

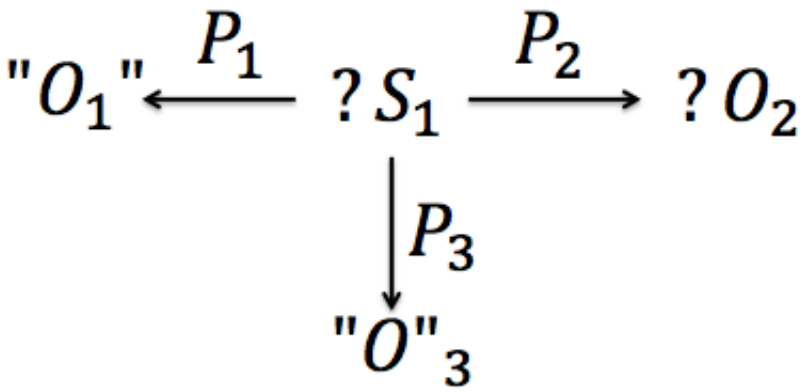
Rule 1: 1-Variable Join

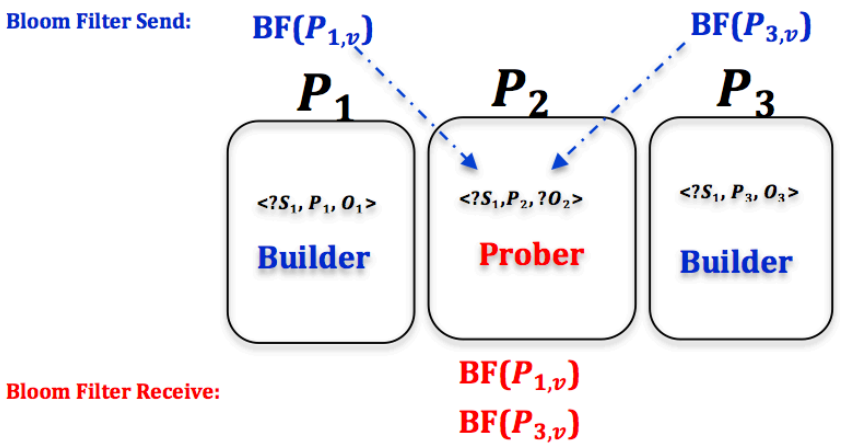




The join of the intermediate results — Structure Based Rules

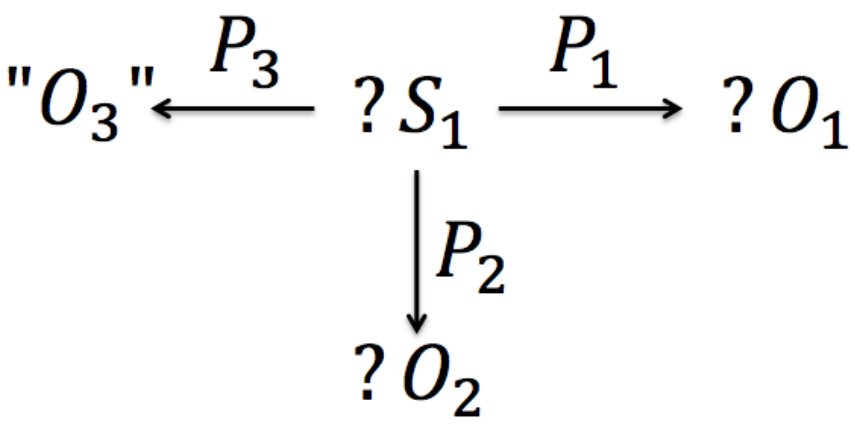
Rule 2: 2-Variable Join

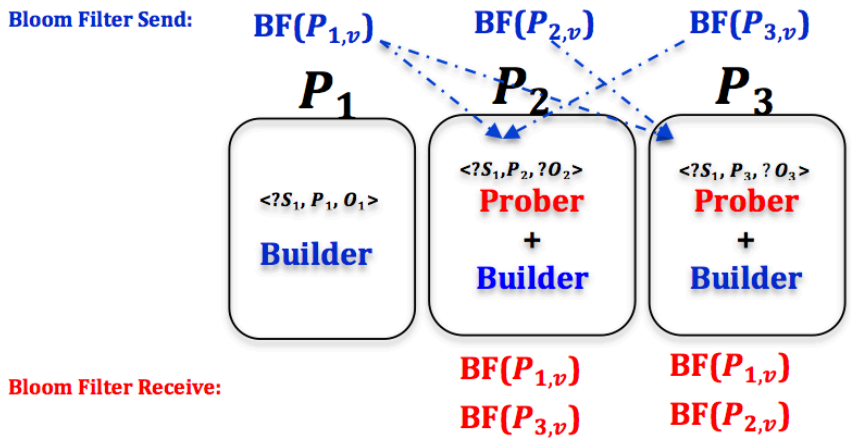




The join of the intermediate results – Structure Based Rules

Rule 3: Multiple-Variable Join





Question: what is the sending and receiving order?

The order of sending and receiving information

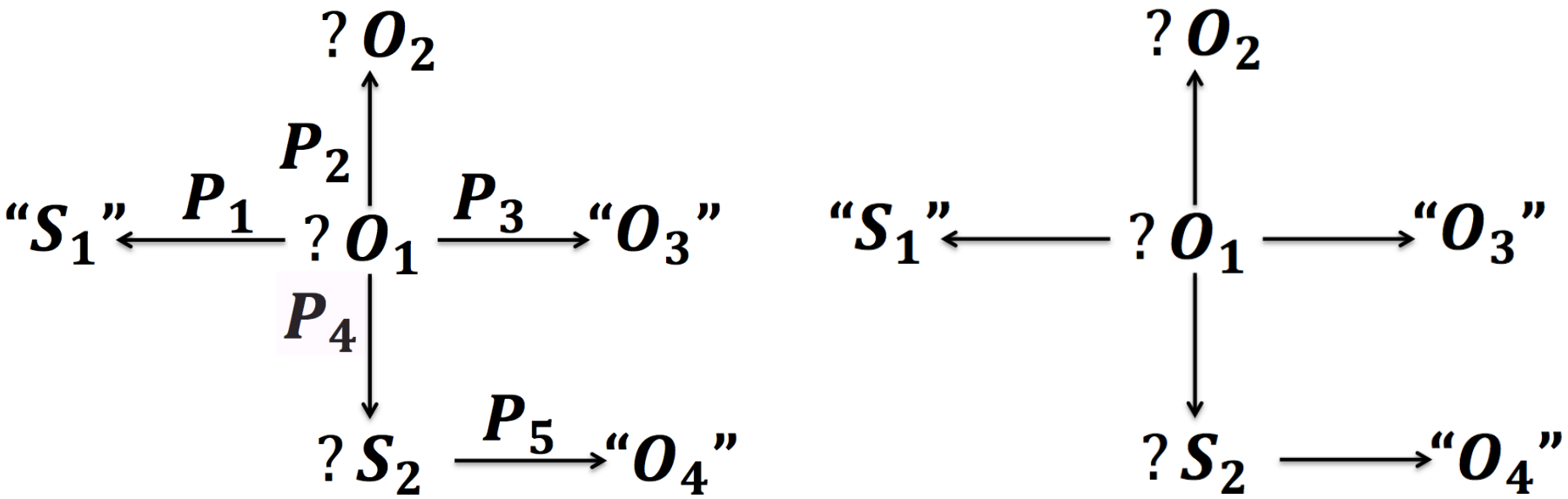
Rule 4: Query Topological Sort

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

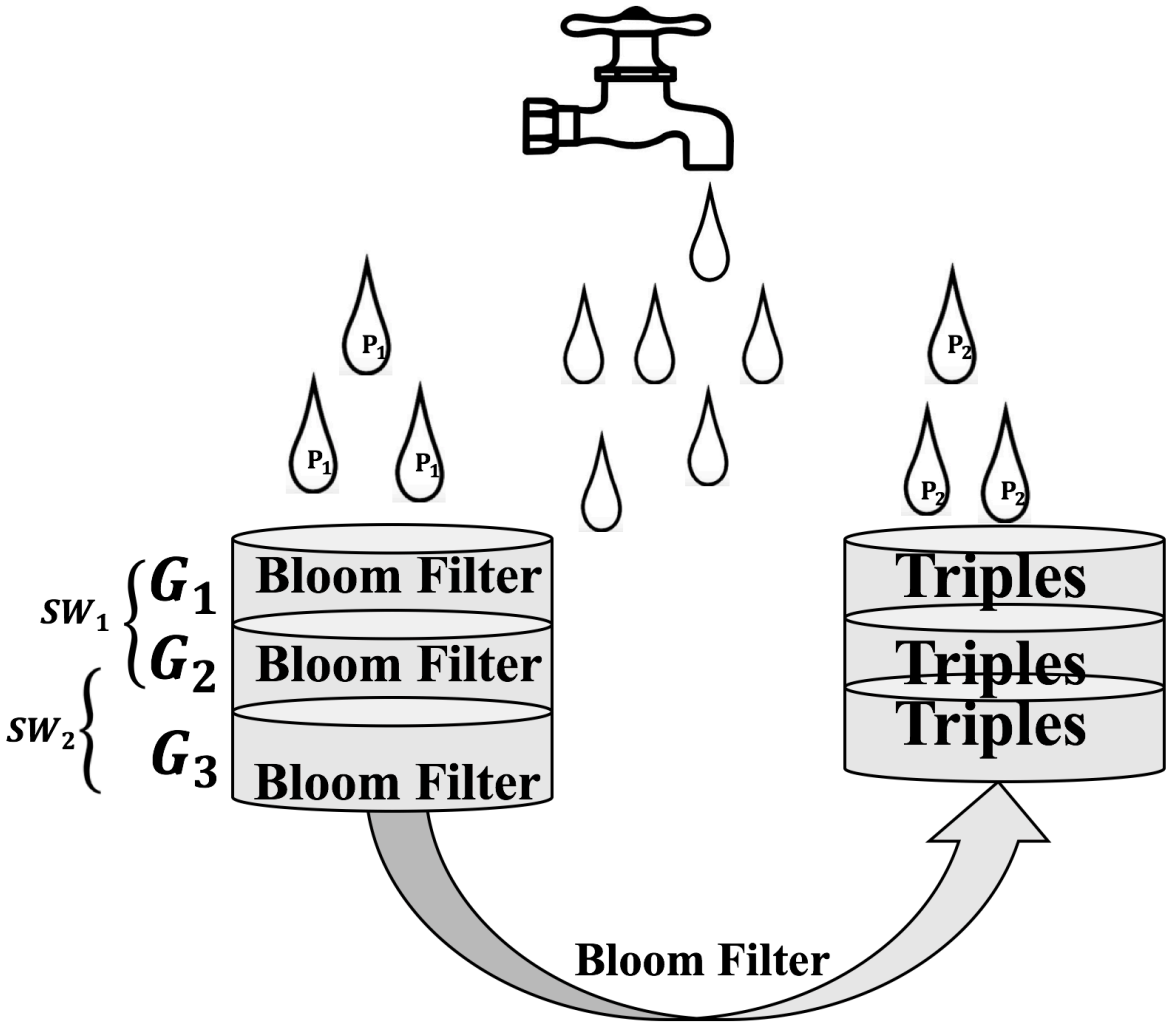


Results: {"O_4", ?S_2, "S_1", "O_3", ?O_2, ?O_1}

Outline

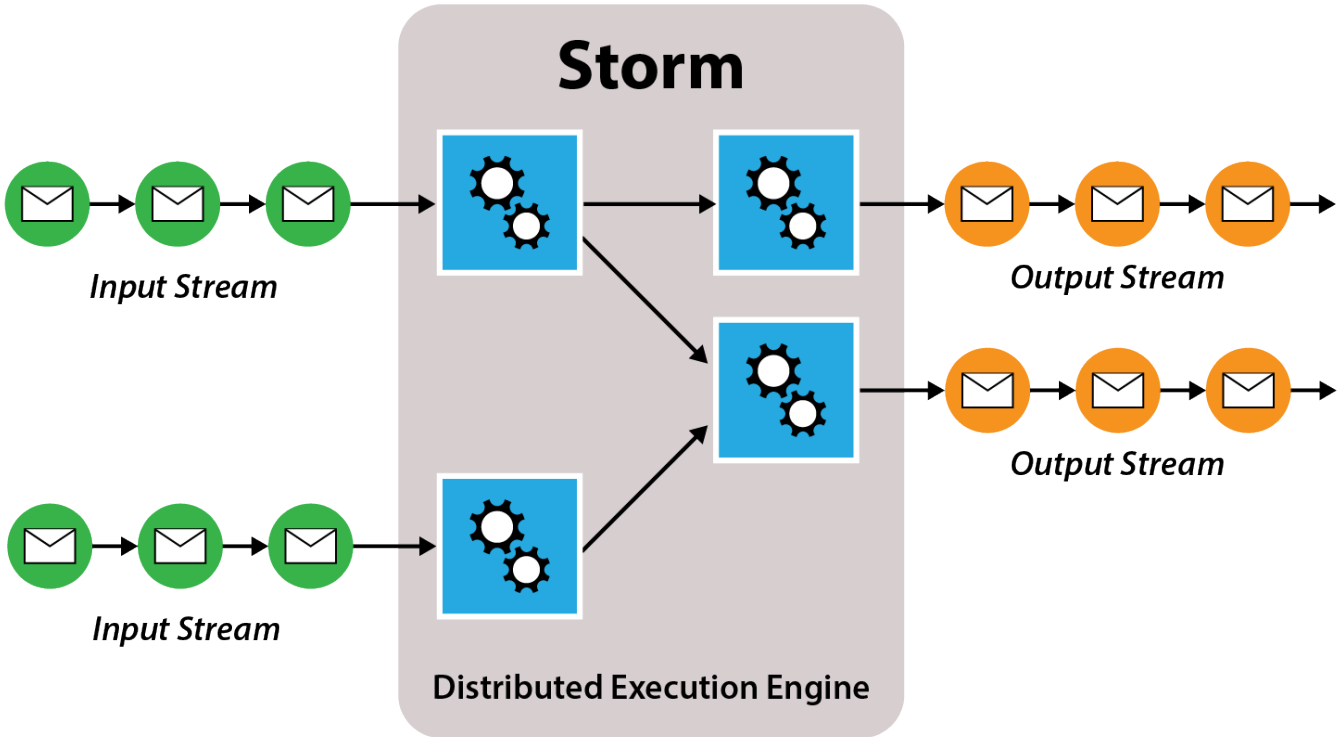
- Introduction
- Query Decomposition and Data Partition
- Parallel and Distributed Query Planner
- Continuous Join
- Implementation
- Experiment Result
- Conclusion

Continuous Join: Sliding Window + Sliding Bloom Filter

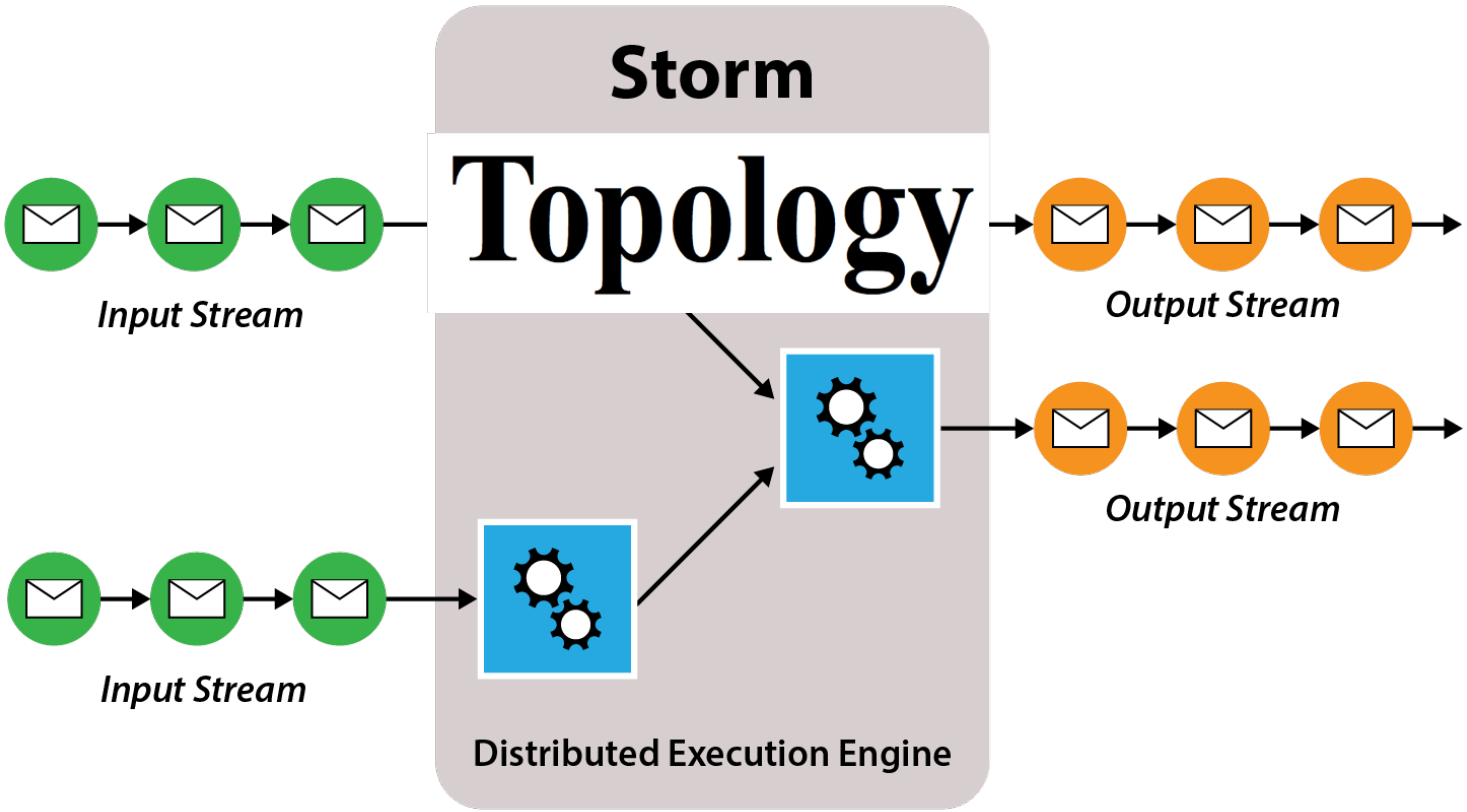


Outline

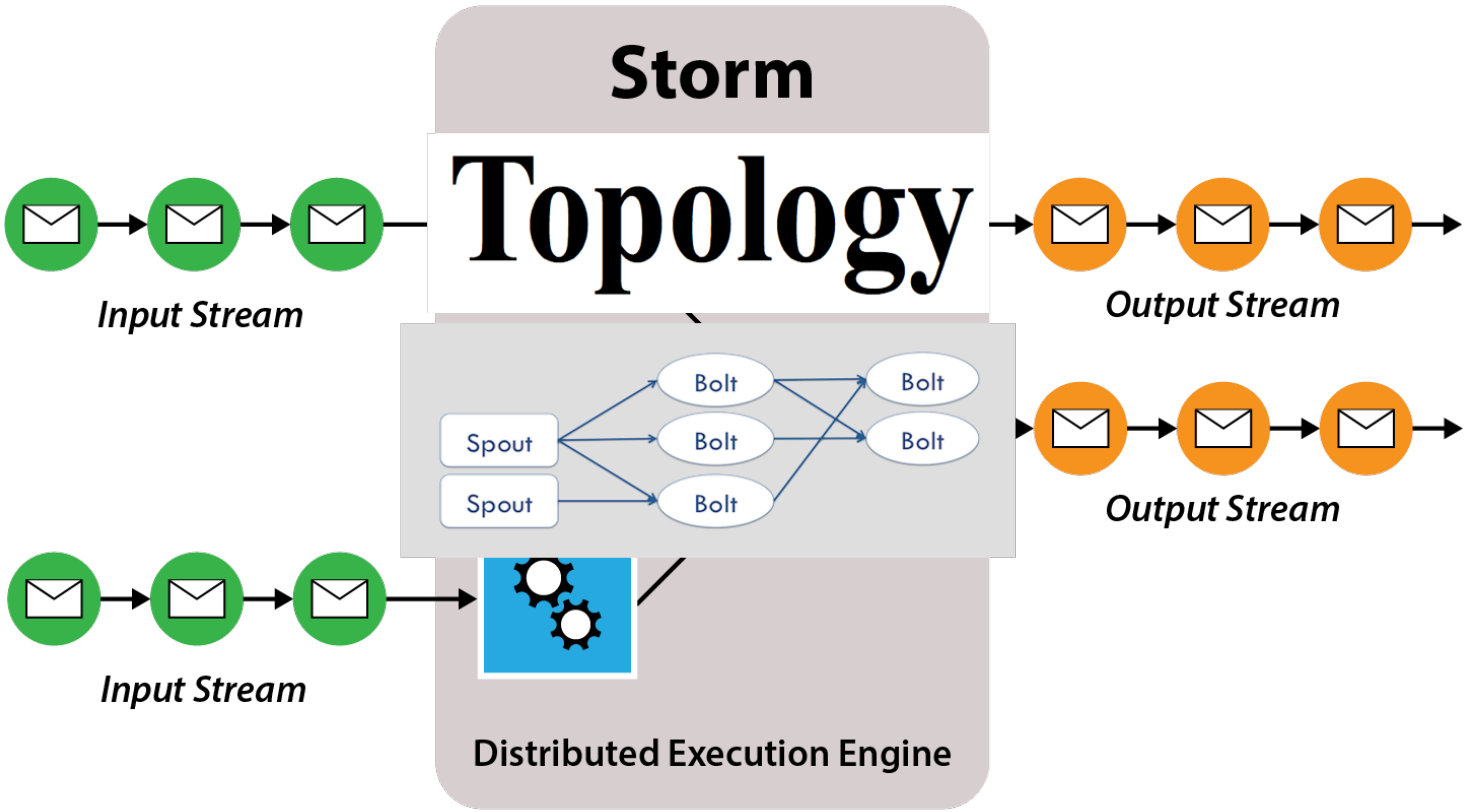
Apache Storm



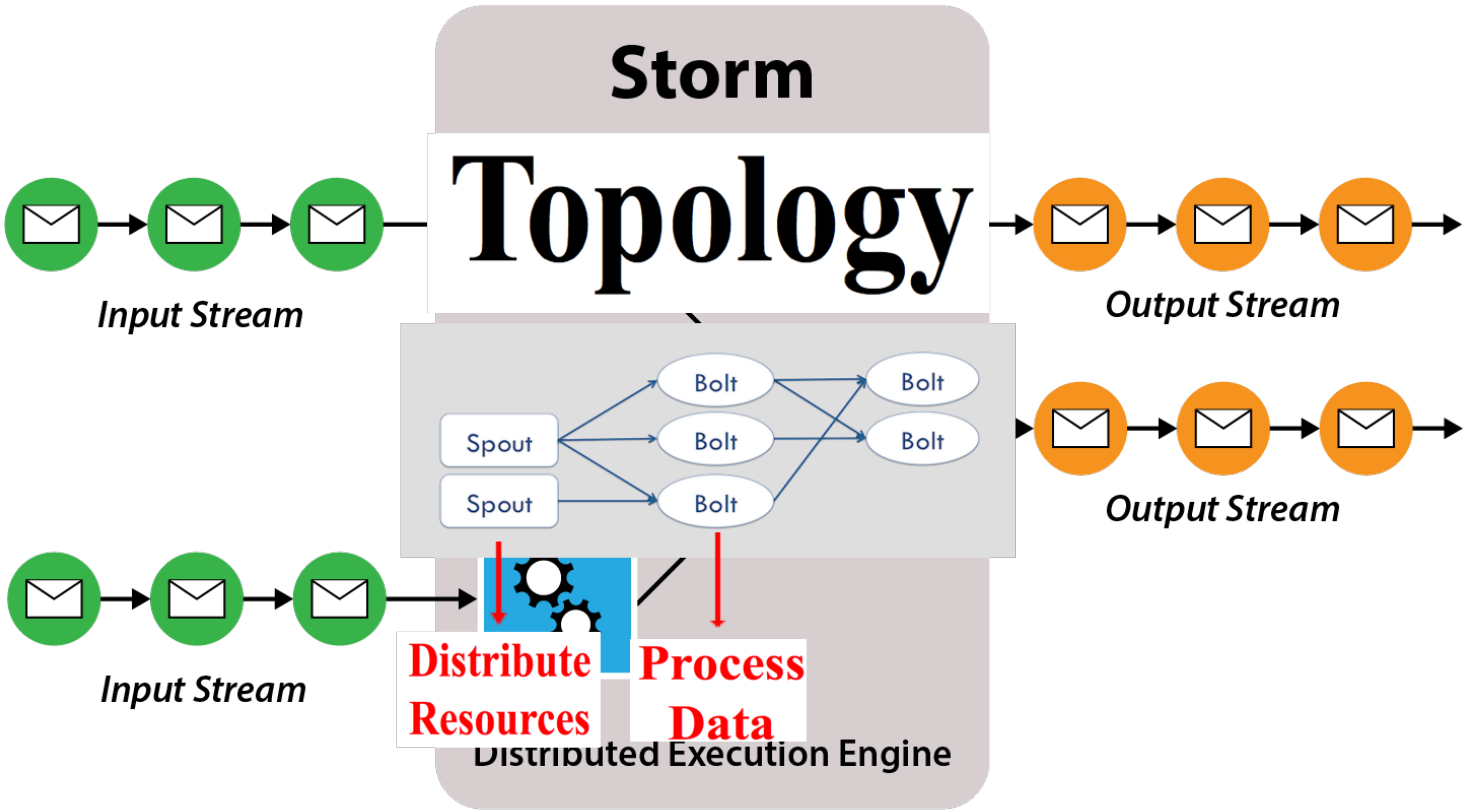
Apache Storm



Apache Storm

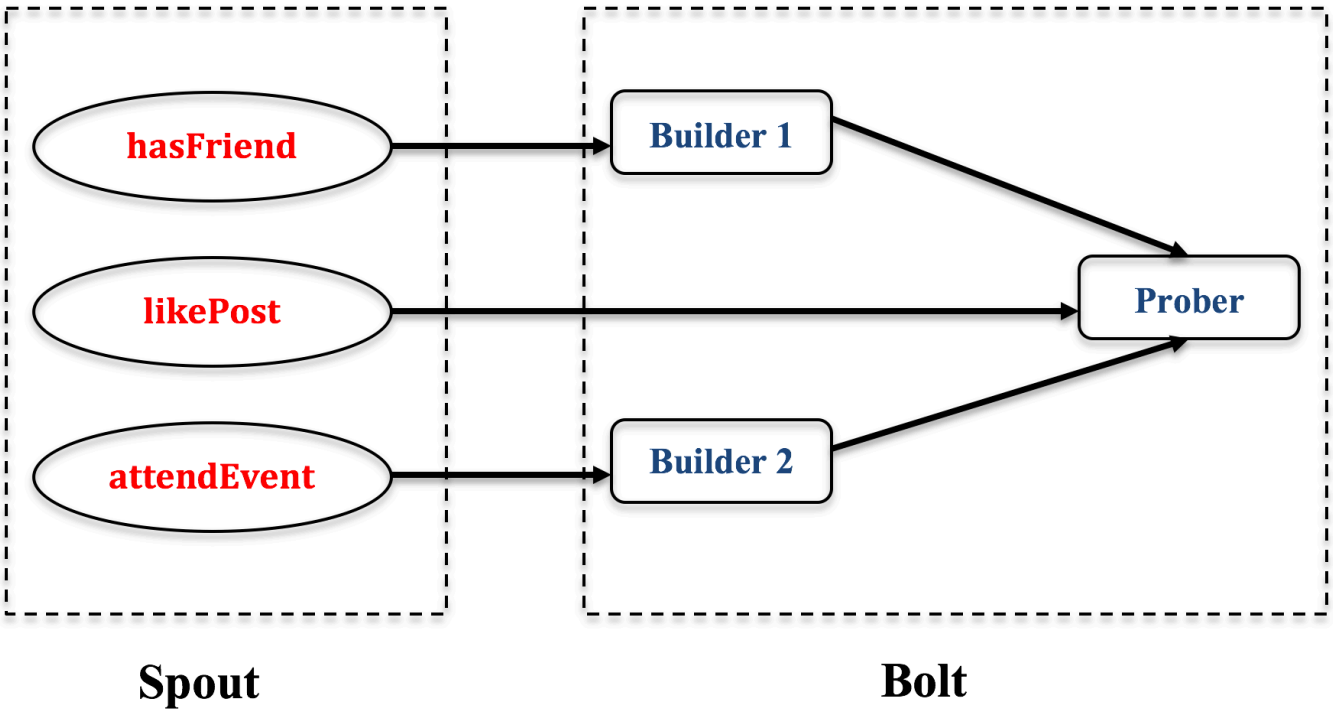


Apache Storm



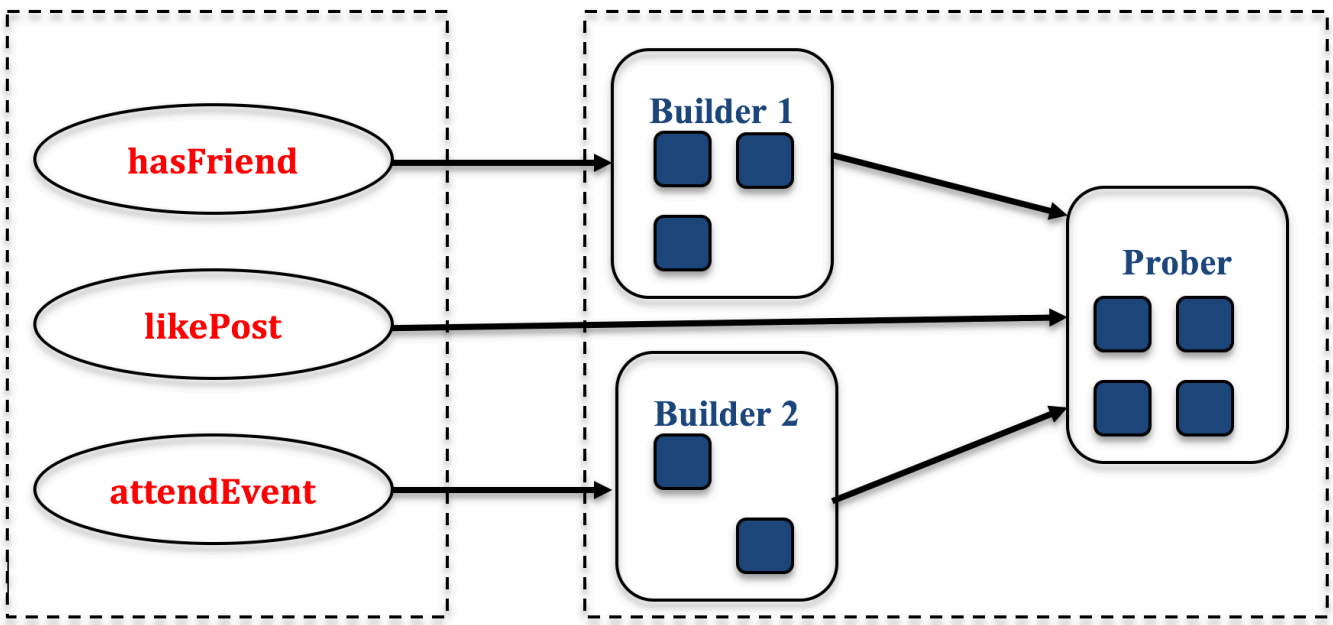
Implementation

```
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.

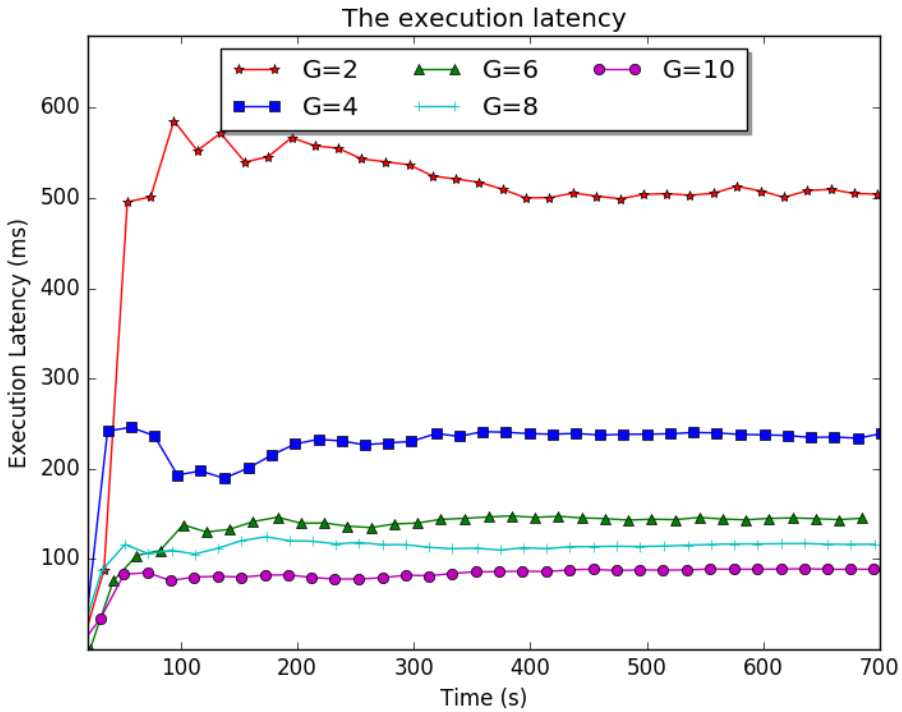
Data sets

- Synthetic data
  - The RDF triples generated in Spouts are distributed to the nodes according to their predicate.
- LUBM Data
  - LUBM has 14 queries, and we have tested query No.1 (1-varialbe join), query No.3 (2-variable join), and query No.4 (multiple-variable join)

Execution Lateny

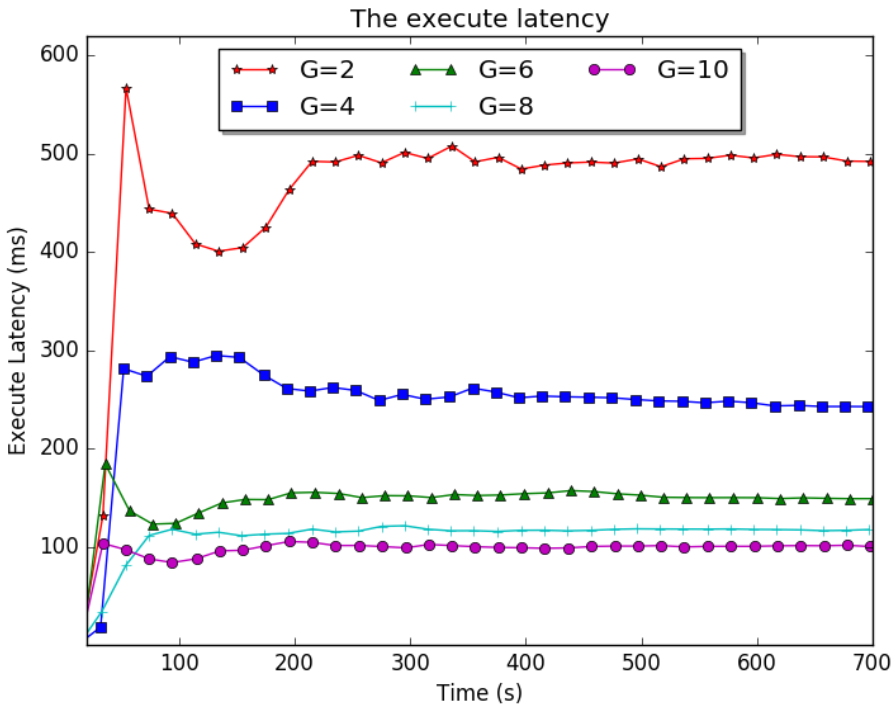
Sliding Window Size = 800 (1-Variable Join)

More Generation $\Rightarrow$ Update more frequently $\Rightarrow$ Faster

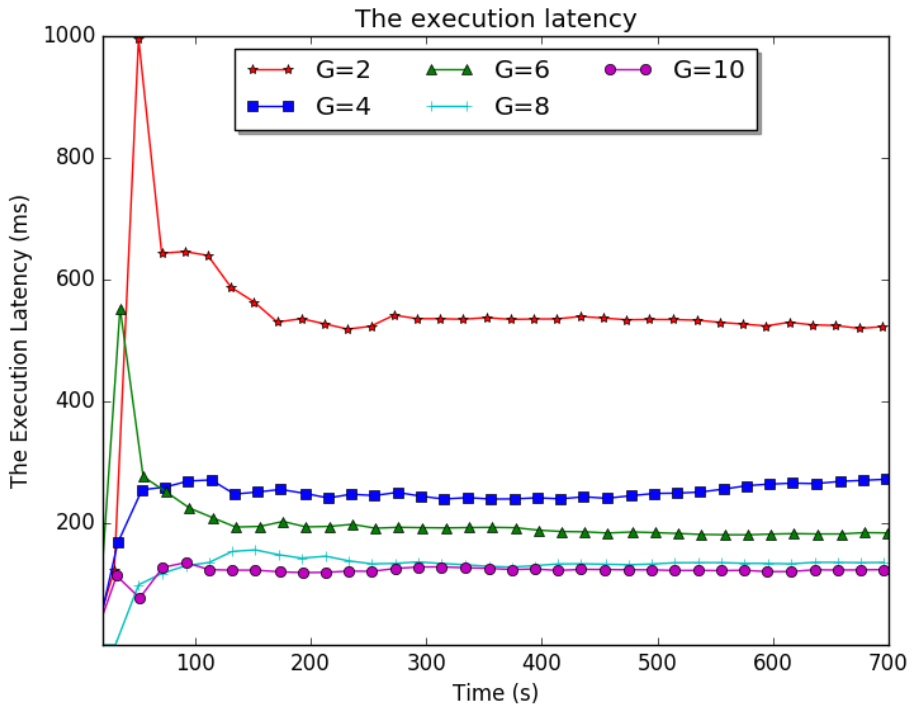


Execution Latency

2-Variable Join

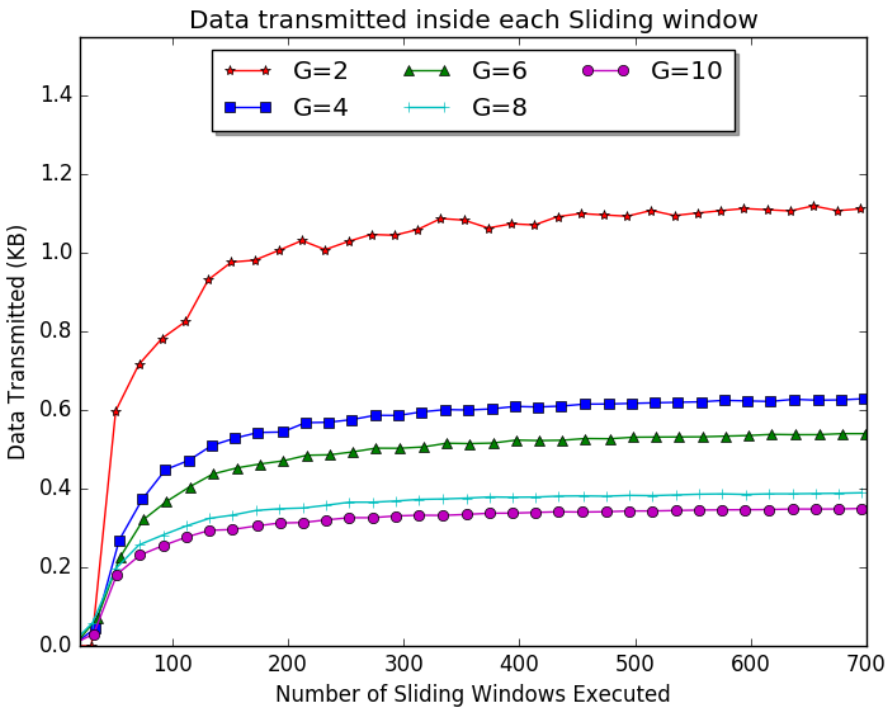


Multiple-Variable Join



Data Transmitted – Sliding Window Size = 800

Multiple-Variable Join



400 more data transmission without Bloom Filters.

Accuracy

We got 100% correct results — Surprise!

The Bloom Filter is designed to contain all the received by the triple pattern, but it only contains the elements which match the triple pattern.

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 SPARQL 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

Publications:

- **K Nearest Neighbour Joins for Big Data on MapReduce: a Theoretical and Experimental Analysis — TKDE 2016 — Ge Song**, J. Rochas, L. Beze, F. Huet, F. Magoulès
- **Solutions for Processing K Nearest Neighbor Joins for Massive Data on MapReduce — PDP 2015 — Ge Song**, J. Rochas, F. Huet, F. Magoulès
- **A Hadoop MapReduce Performance Prediction Method — HPCC 2013 — Ge Song**, Z. Meng, F. Huet, F. Magoulès, L. Yu, X. Lin
- **A Game Theory Based MapReduce Scheduling Algorithm —** Springer New York — **Ge Song**, L. Yu, Z. Meng, X. Lin
- **Detecting topics and overlapping communities in question and answer sites — SNAM 2014 —** Z. Meng, F. Gandon, C.Zucker, **Ge Song**
- **Empirical study on overlapping community detection in question and answer sites — ASONAM 2014 —** Z. Meng, F. Gandon, C. Zucker, **Ge Song**

Thank You!