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

Parallel and Continuous Join Processing for Data Stream

Thèse pour l'obtention du grade de Docteur

Table of Contents

Introduction

Big Data Everywhere

- Google: 24 PB / day
- Facebook: 10 millions 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.
- The cost of network communication in transferring data.
- The dynamic of data — Data Stream

Dynamic Data Stream

Persistent Static Relations $\Rightarrow$ Transient Dynamic Data Streams

Batch-oriented data processing $\Rightarrow$ Real-time stream processing

$\Downarrow$

Architecture Level: possible to add or remove computational nodes based on the current load

Application Level: able to withdraw old results and take new coming data into account

Objective: parallel and continuous processing for Join operation

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

- Data parallelism $\Rightarrow$ Data Driven Join $\Rightarrow$ kNN
- Task parallelism $\Rightarrow$ Query Driven Join $\Rightarrow$ Semantic Join on RDF data

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.

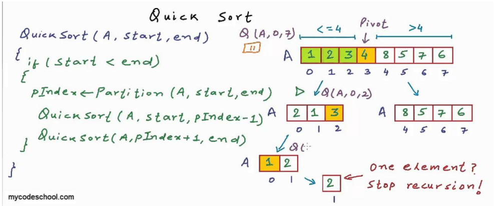
- Query never changes
- Data changes: GPS (2 Dimensions), Twitter (77 Dimensions), Images (128 Dimensions) etc.

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 (Complexity $n \cdot \log(n)$)



Outline

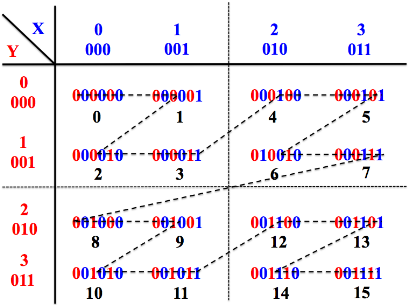
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

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 – To reduce the dimension of data

Space Filling Curve (Z-Value)

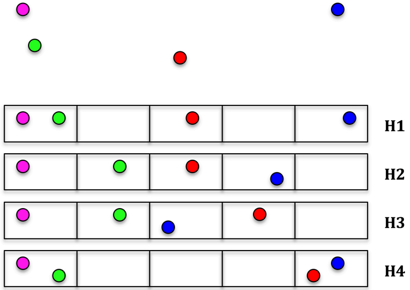


Locality Sensitive Hashing (LSH)

Data Preprocessing – To reduce the dimension of data

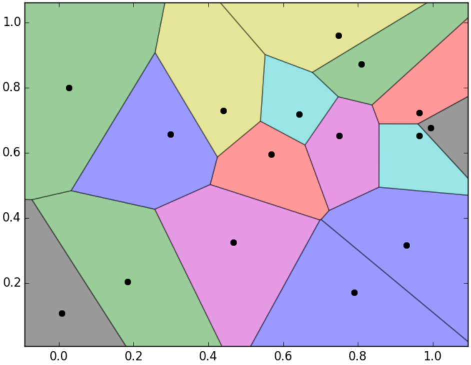
Space Filling Curve (Z-Value)

Locality Sensitive Hashing (LSH)



Data Preprocessing – To select central points (Pivots) of data clusters

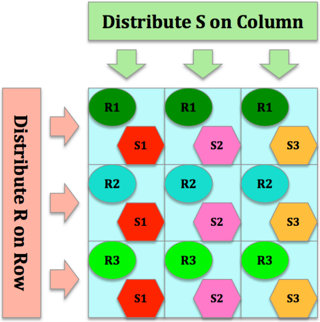
Voronoi Diagram:



Data Preprocessing – To select central points (Pivots) of data clusters

- **Random Selection:** generates a set of samples, calculates the pairwise distance of the points in the sample, and the sample with the biggest sum of distances is chosen as the set of pivots.
- **Furthest Selection:** randomly chooses the first pivot, and calculates the furthest point to this chosen pivot as the second pivot, and so on until having the desired number of pivots.
- **K-Means Selection:** applies the traditional k-means method on a data sample to update the centroid of each cluster as the new pivots in each step, until the set of pivots stabilizes.

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.

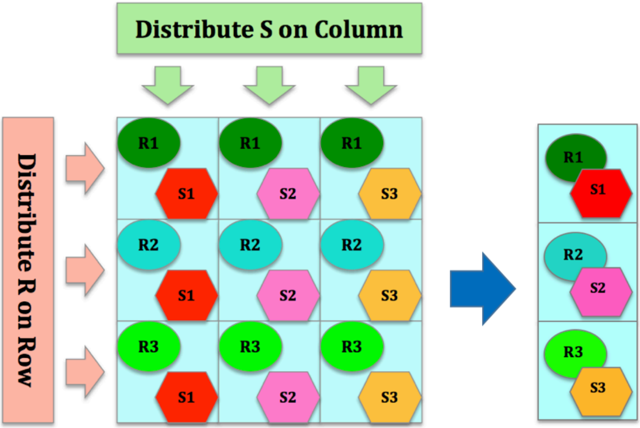
More precisely, what we want is: for every partition R_i ($\cup_i R_i = R$), find a corresponding partition S_j ($\cup_j S_j = S$), where:

$$kNN(R_i \ltimes S) = kNN(R_i \ltimes S_j)$$

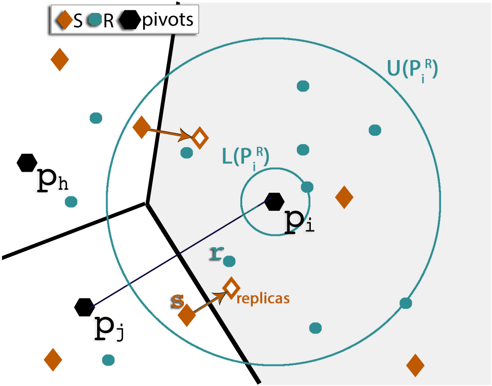
And,

$$kNN(R \ltimes S) = \bigcup kNN(R_i \ltimes S_j)$$

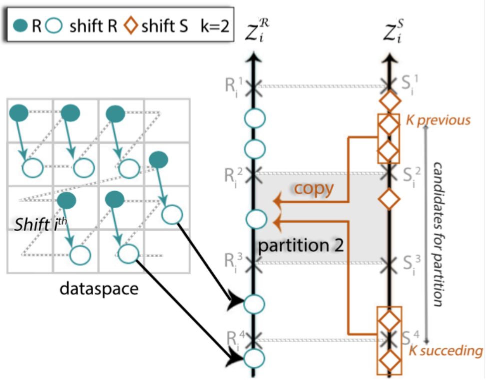
Data Partitioning – Motivation



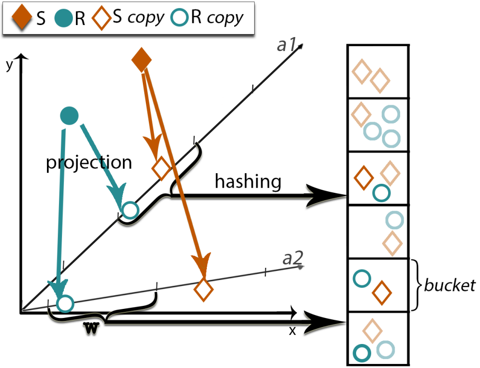
Data Partitioning – Distance Based Partitioning Strategy



Data Partitioning – Size Based Partitioning Strategy – Z-Value



Data Partitioning – Size Based Partitioning Strategy – LSH



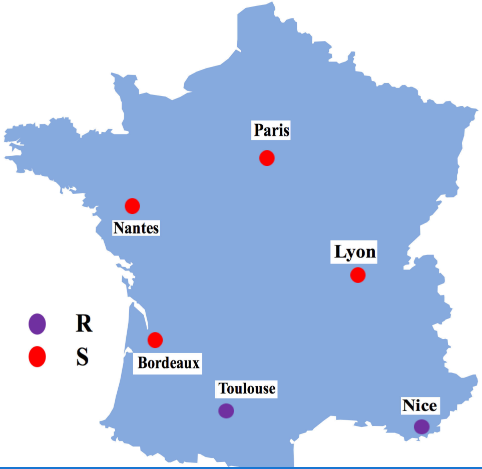
Computation

- One job — Direct 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.

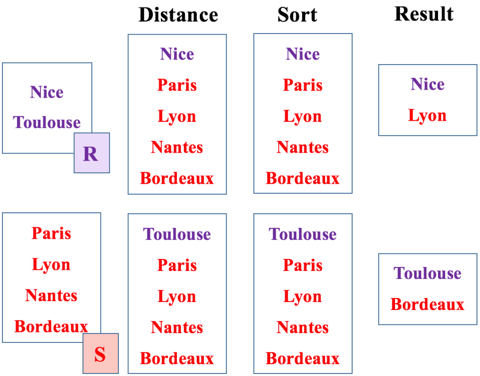
Computation – Example

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



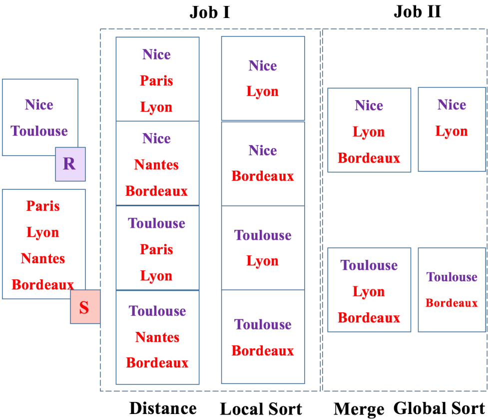
Computation – Example

One Job:

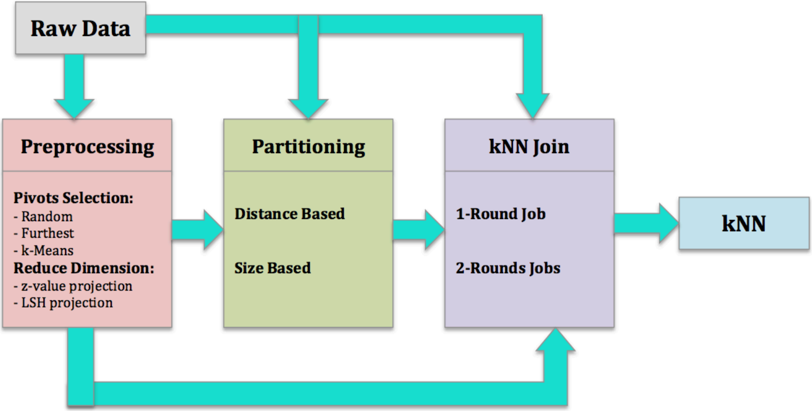


Computation – Example

Two Jobs:



Workflow


Raw Data
Preprocessing
Pivots Selection:
- Random
- Furthest
- k-Means
Reduce Dimension:
- z-value projection
- LSH projection
Partitioning
Distance Based
Size Based
kNN Join
1-Round Job
2-Rounds Jobs
kNN

Outline

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

Theoretical Analysis

- Load Balance
- Accuracy
- Complexity

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:

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

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

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

$n \ll |S| \Rightarrow |R_i| = |R_j|$ 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
  - 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$(R_i \ltimes S)$:** the larger this number is, the more information is exchanged through the network.
- **The number of final candidates for each object r_i:** We have seen that advanced algorithms use pre-processing and partitioning techniques to reduce this number as much as possible.

Main Overhead

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

Outline

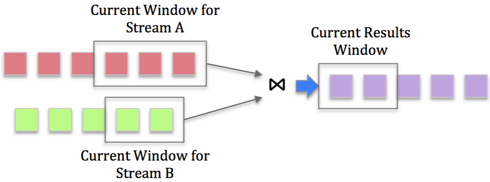
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

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



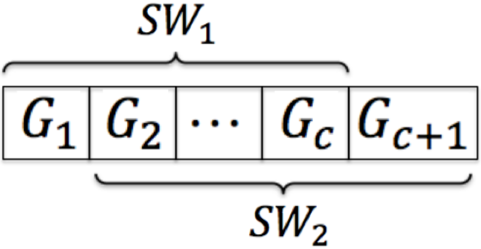
Sliding Window – Two Strategies

- **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 Expiration 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 Re-execution 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)

		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)

Purpose: partition the new data items without moving the old ones.

Naive Bayes Theory

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

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{1}$$

DRDS – Advanced Method – Naive Bayes Partitioning

We consider the n partitions from N_1 to N_n as n different classes and the already partitioned data as training set. The probability that a new point x belongs to N_i is:

$$P(N_i|x) = \frac{P(x|N_i) \cdot P(N_i)}{P(x)} \tag{2}$$

And x should be assigned to the partition N_y which has the biggest probability:

$$x \in N_y,\ where\ P(N_y|x) = max\{P(N_1|x), P(N_2|x), ..., P(N_n|x)\} \tag{3}$$

Naive Bayes Partitioning

Partitioning Problem = The calculation of $P(N_i|x)$

$$P(N_i|x) = P(N_i|(l_1(x), l_2(x), ..., l_p(x))) \tag{4}$$

According to Bayes' Theorem:

$$= \frac{P((l_1(x), l_2(x), ..., l_p(x))|N_i)}{P(l_1(x), l_2(x), ..., l_p(x))} \tag{5}$$

Since l_1, l_2, ..., l_p are independent, we have:

$$= \frac{P(l_1(x)|N_i) \cdot P(l_2(x)|N_i) ... \cdot P(l_p(x)|N_i) \cdot P(N_i)}{P(l_1(x)) \cdot P(l_2(x)) ... \cdot P(l_p(x))} \tag{6}$$

$P(l_j(x)|N_i)$ is the probability of the appearance of $l_j(x)$ on N_i, and this probability is decided by the distribution of data on N_i.

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

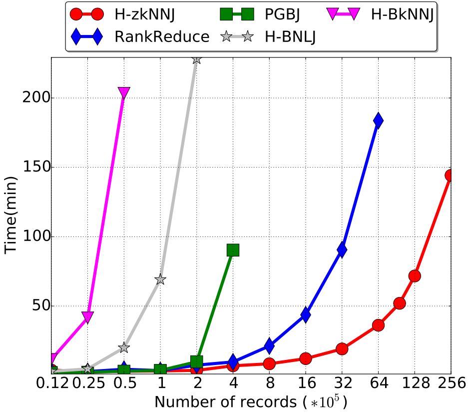
Measures: (y axis)

- Execution Time
- Communication Overhead
- Recall and Precision

Practical Analysis

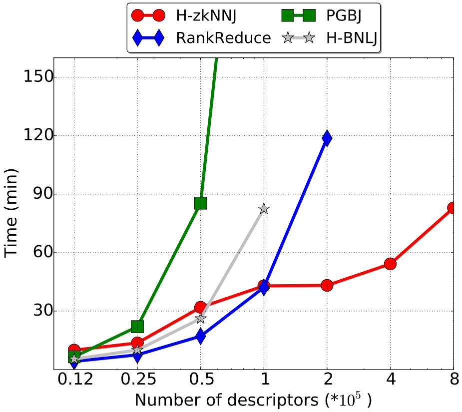
Evaluation Result – Verify the theoretical Analysis

Execution Time for Geo dataset (2 dimensions):



Evaluation Result — Surprise

Execution Time for Image dataset (128 dimensions):



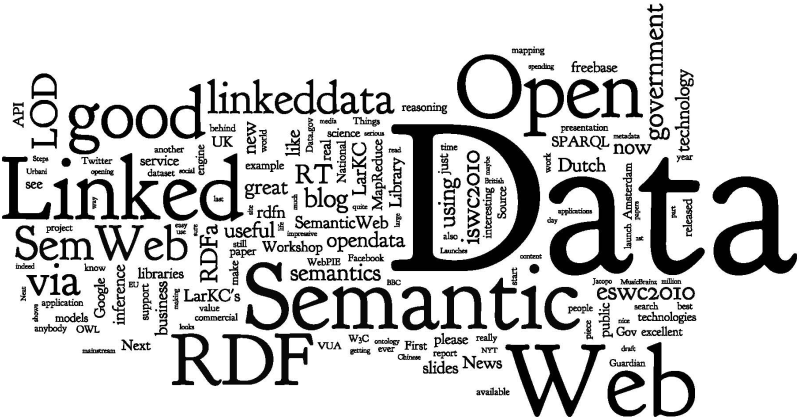
Outline

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

Conclusion

Algorithm	Advantage	Shortcoming	Typical Usecase
H-BkNNJ	Trivial to implement	1. Breaks very quickly 2. Optimal parallelism difficult to achieve a priori	Any tiny and low dimension dataset (~ 25000 records)
H-BNLJ	Easy to implement	1. Slow 2. Very large communication overhead	Any small/medium dataset (~ 100000 records)
PGBJ	1. Exact solution 2. Lowest disk usage 3. No impact on communication overhead with the increase of k	1. Cannot finish in reasonable time for large datasets 2. Poor performance for high dimension data 3. Large communication overhead 4. Performance highly depends on the quality of a priori chosen pivots	1. Medium/large dataset for low/medium dimension 2. Exact results
H-zkNNJ	1. Fast 2. Does not require a priori parameter tuning 3. More precise for large k 4. Always give the right number of k	1. High disk usage 2. Slow for large dimension 3. Very high space requirement ratio for small values of k	1. Large dataset of small dimension 2. High values of k 3. Approximate results
RankReduce	1. Fast 2. Low footprint on disk usage	1. Fine parameter tuning required with experimental set up 2. Multiple hash functions needed for acceptable recall 3. Different quality metrics to consider (recall + precision)	1. Large dataset of any dimension 2. Approximate results 3. Room for parameter tuning

Part II: Query Driven Stream Join (RDF)



Outline

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

Outline

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

Introduction – RDF Data Model

- **R**esource **D**escription **F**ramework 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

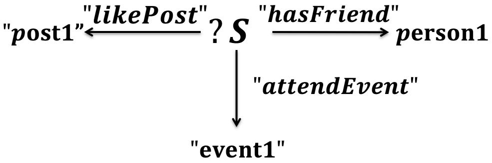
- SPARQL is a W3C recommendation query language for querying RDF data.
- The basic component of a SPARQL query is the triple patterns.
- A triple pattern is a special kind of triple where S, P and O can be either a literal or a variable.

An Example (Triple Pattern Representation):

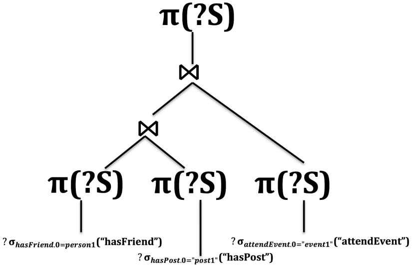
```
SELECT ?S
WHERE{
  Q1     ?S “hasFriend” person1 .
  Q2     ?S “likePost” “post1” .
  Q3     ?S “attendEvent” “event1”
}
```

Introduction – SPARQL Query Example

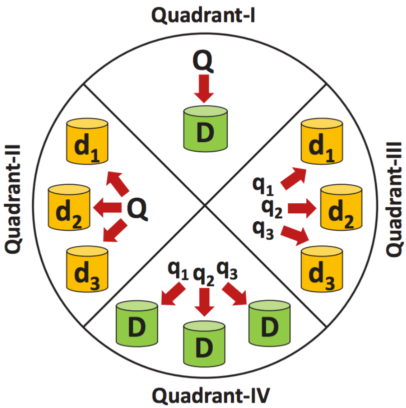
Graph Representation:



Relational Representation:



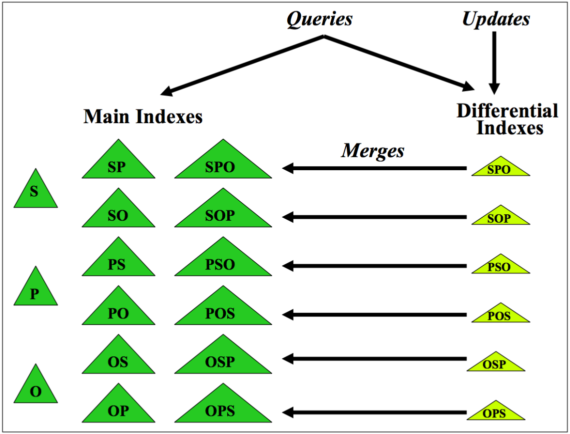
Related Works – 4 Types of Processing



Partitioning Strategies

- 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



Steps

- Partition the RDF streams, and disperse these sub-streams to the 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
- Analysis
- Implementation
- Experiment Result
- Conclusion

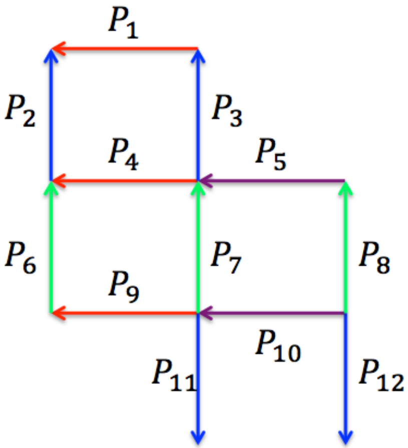
Query Decomposition

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

- It is simple, and does not require any complicated computations.
- The sub-streams can be easily assigned to the sub-queries.
- The performance or accuracy of this method does not depend on the index or replication of data.
- Among the triples processed by a query, the number of different subjects or objects involved could be tens of thousands; But the number of triple patterns is a limited and fixed number.

Sub-Query Scheduling

Edge Coloring Method:



Data Partitioning

Partitioning Strategy: The triples with the same predicate will be assigned to the same nodes that hold the triple pattern with that 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.
- The number of different types of subjects and objects involved could be tens of thousands; But the number of different predicates is limited, and this number must be smaller than the number of triple patterns that compose this query.

Outline

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

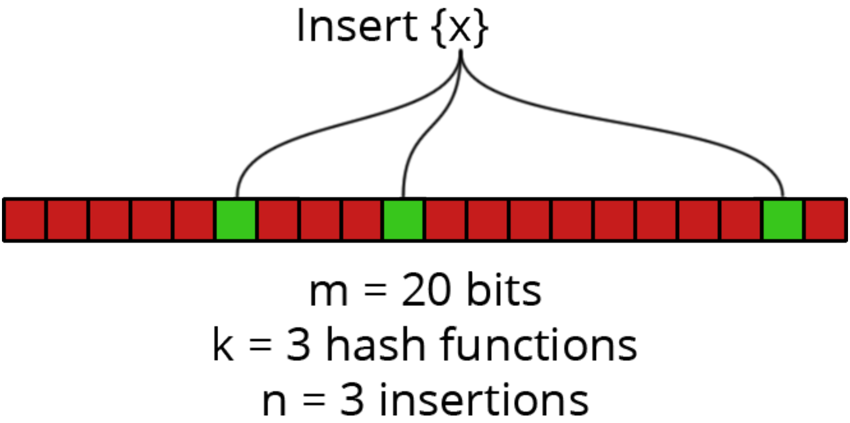
Problems to be solved to complete the decomposition and partitioning idea

- The communication among nodes
- The join of the intermediate results produced by each triple pattern
- The order of sending and receiving information

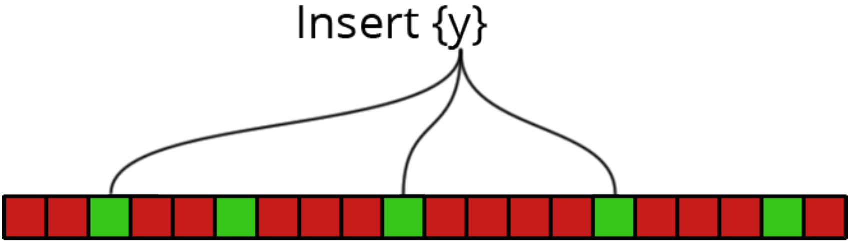
The communication among nodes – Bloom Filter

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

The communication among nodes – Bloom Filter

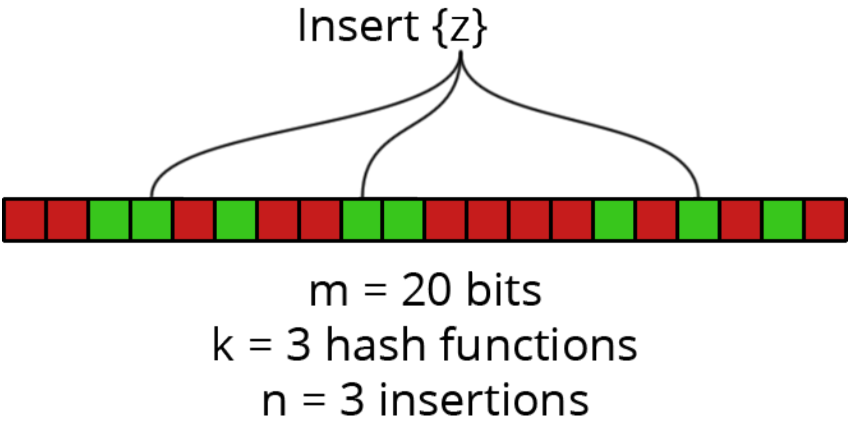


The communication among nodes – Bloom Filter

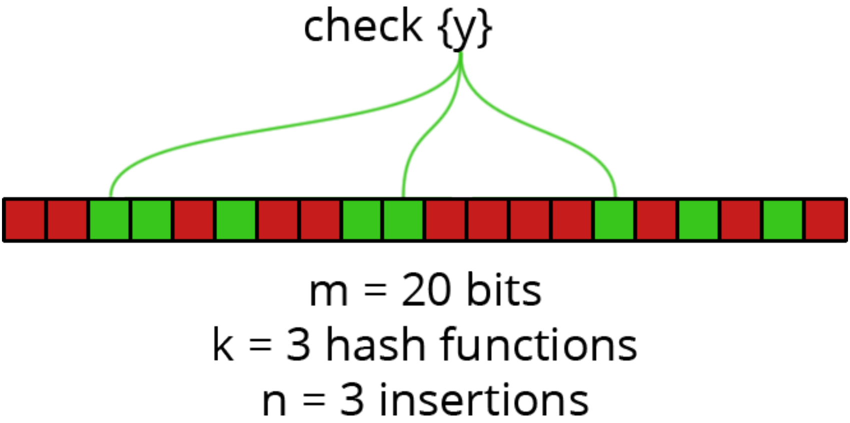


m = 20 bits
k = 3 hash functions
n = 3 insertions

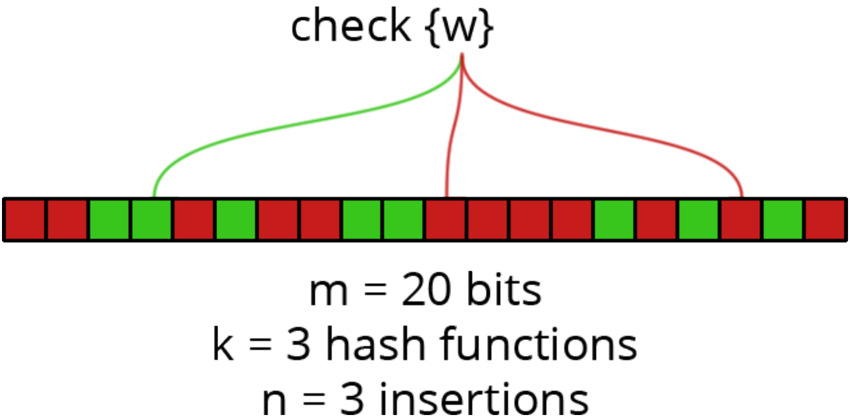
The communication among nodes – Bloom Filter



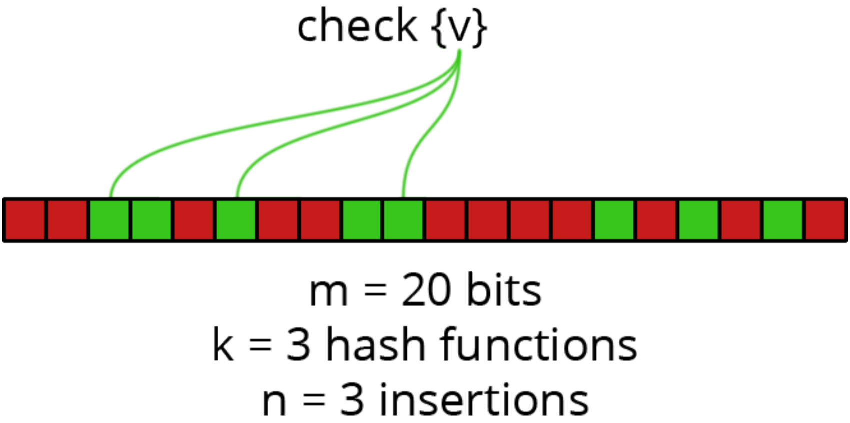
The communication among nodes — Bloom Filter



The communication among nodes — Bloom Filter



The communication among nodes – Bloom Filter



The communication among nodes – Bloom Filter

Minimize false positive risk:
$k = (m/n) \ln 2$

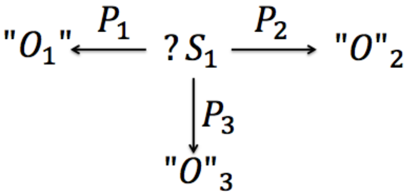
m = 20 bits
k = 3 hash functions
n = 3 insertions

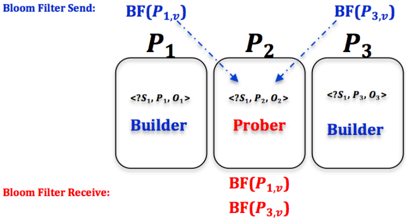
Bloom Filter – Build and Probe

- **Builder:** The triple patterns used to Build the Bloom Filter
- **Prober:** The triple patterns used to Probe the Bloom Filter

The join of the intermediate results – Structure Based Rules

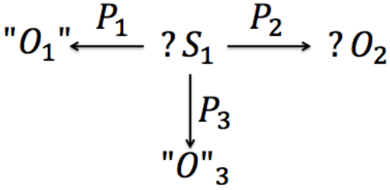
Rule 1: 1-Variable Join

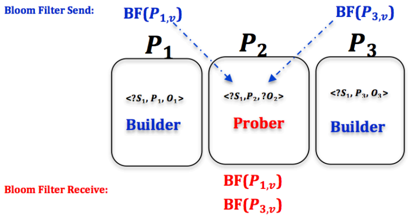




The join of the intermediate results – Structure Based Rules

Rule 2: 2-Variable Join

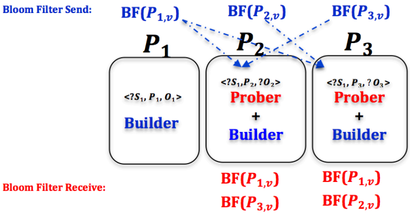




The join of the intermediate results – Structure Based Rules

Rule 3: Multiple-Variable Join

$$\begin{tikzcd} \text{"}O_3\text{"} & ?S_1 \arrow[l, "P_3"'] \arrow[r, "P_1"] \arrow[d, "P_2"] & ?O_1 \\ & ?O_2 & \end{tikzcd}$$



The order of sending and receiving information

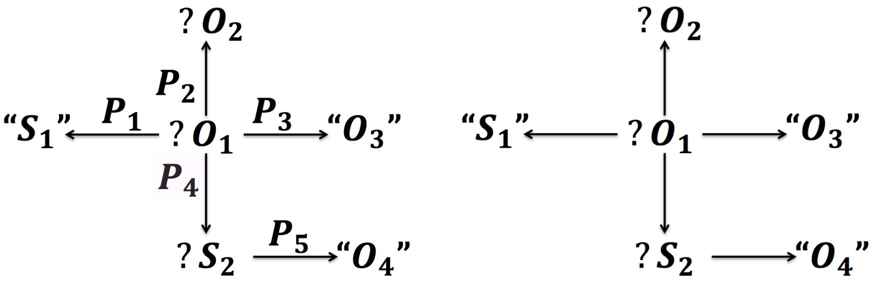
Rule 4: Query Topological Sort

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

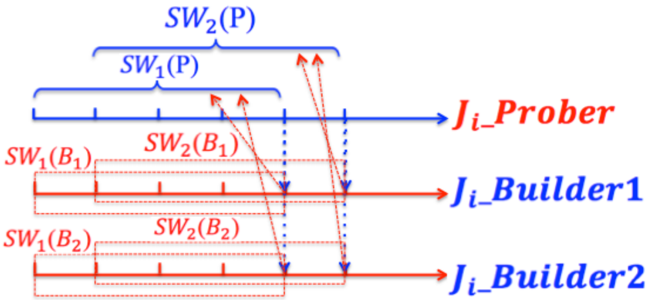


Results: {"O_4", $?S_2$, "S_1", "O_3", $?O_2$, $?O_1$}

Outline

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

Analysis

- Bloom Filters
- Dominating Parameters
- Complexities

Analysis About Bloom Filters — False Positive Rate

Theorem 1

Suppose that we use k hash functions to insert n elements into an m bits Bloom Filter, then the false positive rate p for a standard Bloom Filter is a function of n, m and k, and

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

Idea: Balls and Bins

Analysis About Bloom Filters — Minimum False Positive Rate

Theorem 2

The false positive p reaches the minimum value, when

$$e^{-\frac{nk}{m}} = \frac{1}{2}$$

At this extreme point,

$$k = \ln 2 \times \frac{m}{n}$$

And,

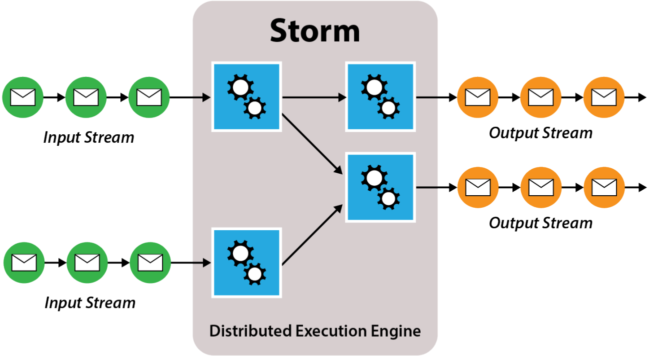
$$p = \frac{1}{2}^{k} = 2^{-\ln 2 \times \frac{m}{n}}$$

Idea: p reaches the minimum value, when the derivation of p reaches 0.

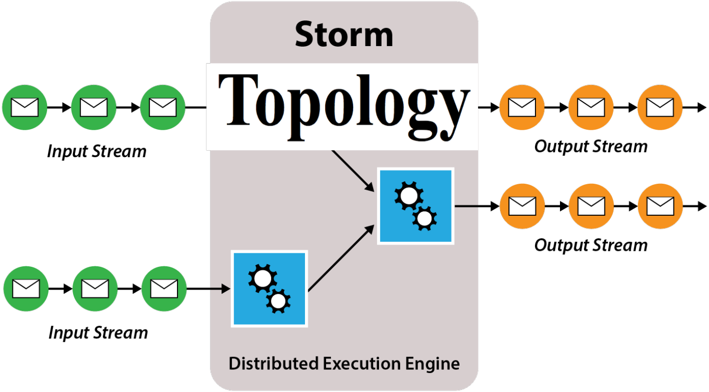
Outline

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

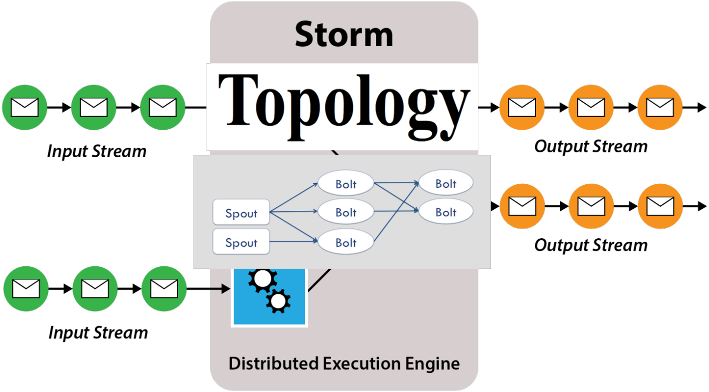
Apache Storm



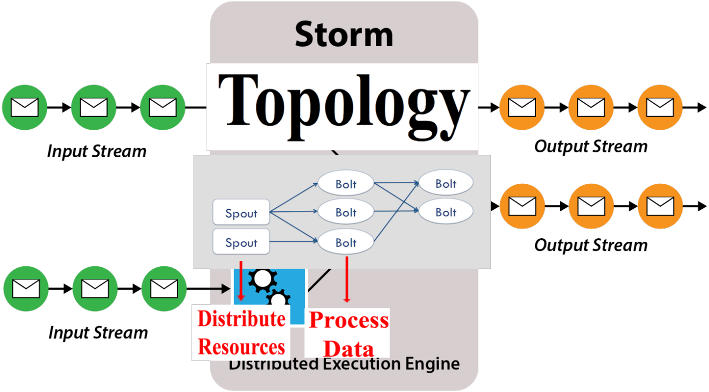
Apache Storm



Apache Storm

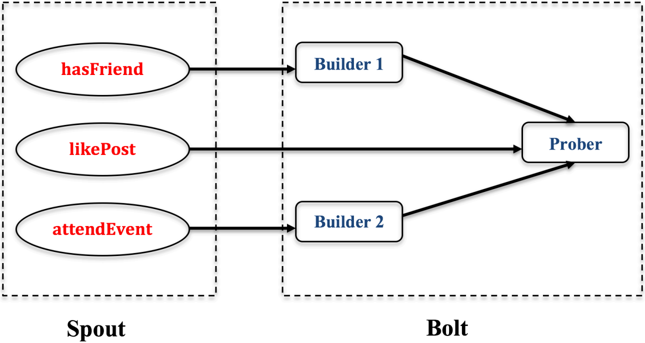


Apache Storm



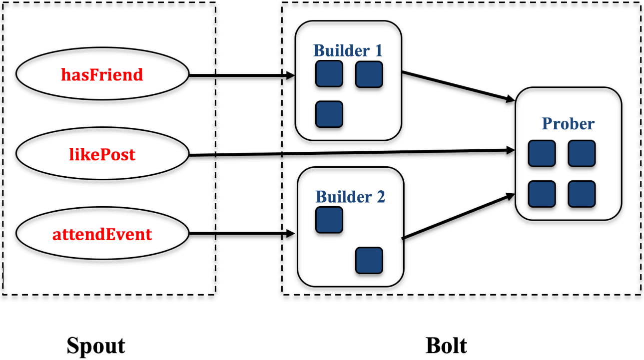
Implementation

```
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
- Analysis
- Implementation
- Experiment Result
- Conclusion

Experiment Setting

- We evaluate the system on Grid 5000 4, with 11 nodes. Among them, one is reserved to be Nimbus, and the rest 10 nodes are used for computing.
- 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)

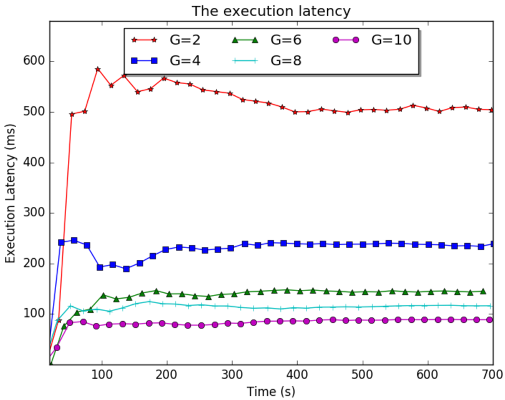
- Sliding Window Size
- Number of Generations

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

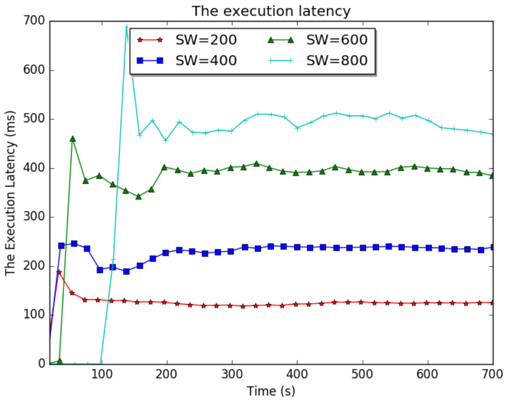
Execution Lateny

Sliding Window Size = 800 (1V)



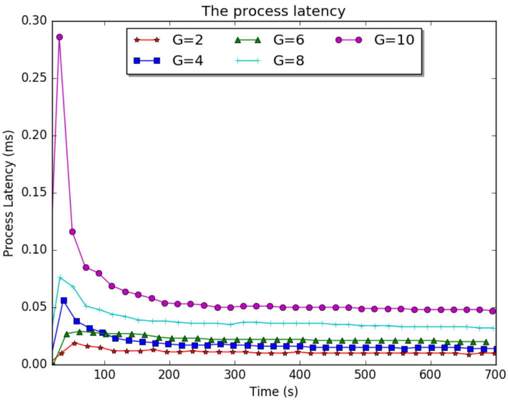
Execution Latency

Number of Generation = 6 (1V)



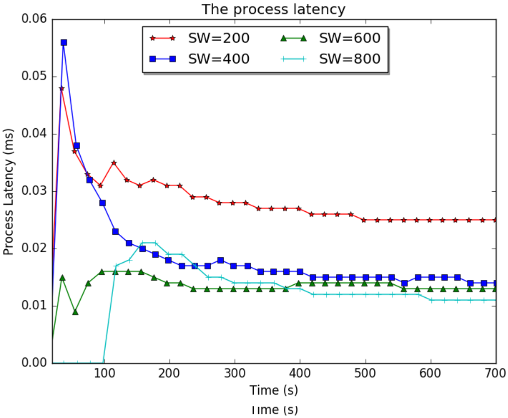
Processing Lateny

Sliding Window Size = 800 (1V)



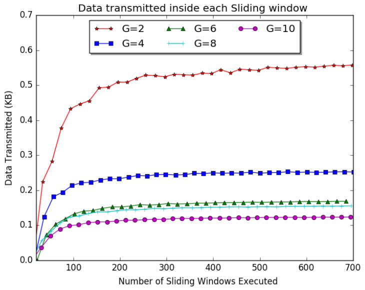
Processing Latency

Number of Generation = 6 (1V)

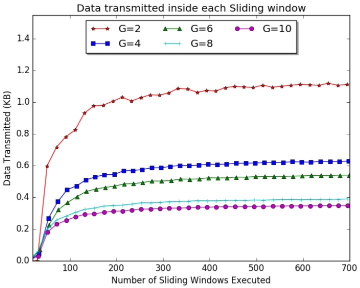


Data Transmitted – Sliding Window Size = 800

1V



MV



Accuracy

We got 100% correct results — Surprise!

- use p (false positive rate) and n (number of element 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
- Analysis
- Implementation
- Experiment Result
- Conclusion

Conclusion

- Query Decomposition and Data Partition
  - According to predicate
- Parallel and Distributed Query Planner
  - use Bloom Filter to communicate among sub-queries
  - 3 types of different kinds of joins according to the structure
  - one rule for communication order
- Continuous Join
  - Sliding Window + Sliding Bloom Filter
- Analysis
  - Bloom Filters
  - Dominating Parameters for the System
- Implementation
  - Spout + BuilderBolt + ProberBolt
- Experiment Result
  - Processing Latency + Execution Latency + Data Transmission

Conclusion and Future Work

Conclusion

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

Future Work – Research Part

- Enrich the advanced re-partitioning strategy for kNN streaming join (theoretical and experimental analysis and proof)
- Enrich the query engine by providing operations such as FILTER, OPTIONAL, etc.

Future Work – Use Cases

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

Thank You!