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 dynamics of data.

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

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 $\mathbb{S}$.

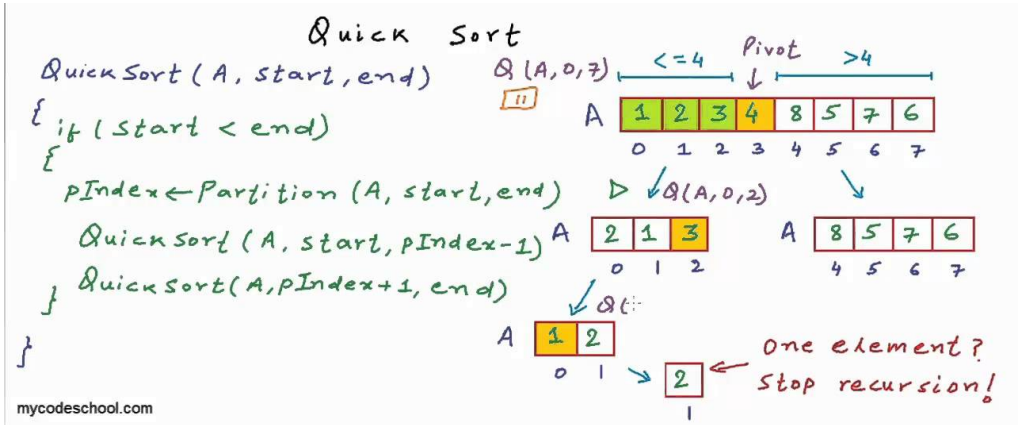
- Query never changes
- Data changes: GPS (2 Dimensions), Twitter (77 Dimensions), Images (128 Dimensions) etc.

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



Outline

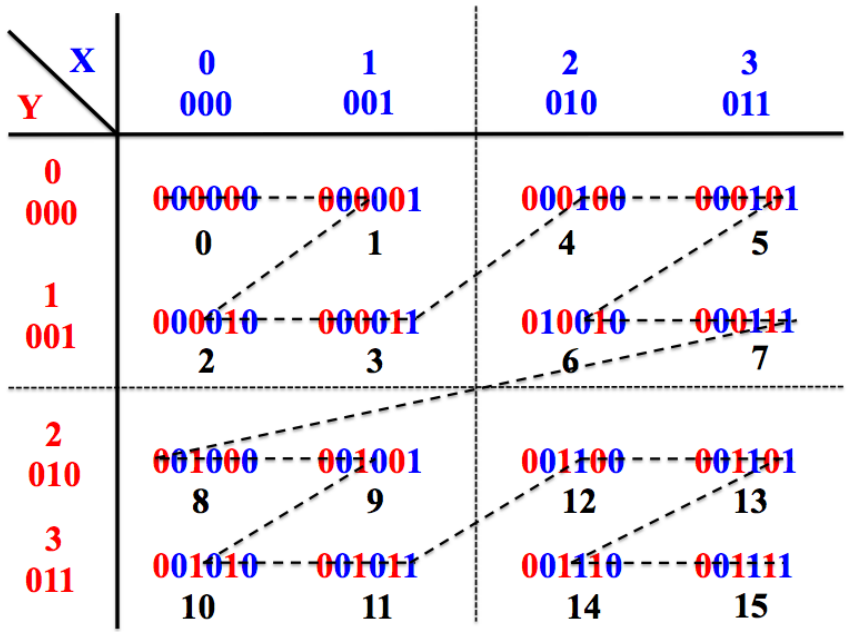
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

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)

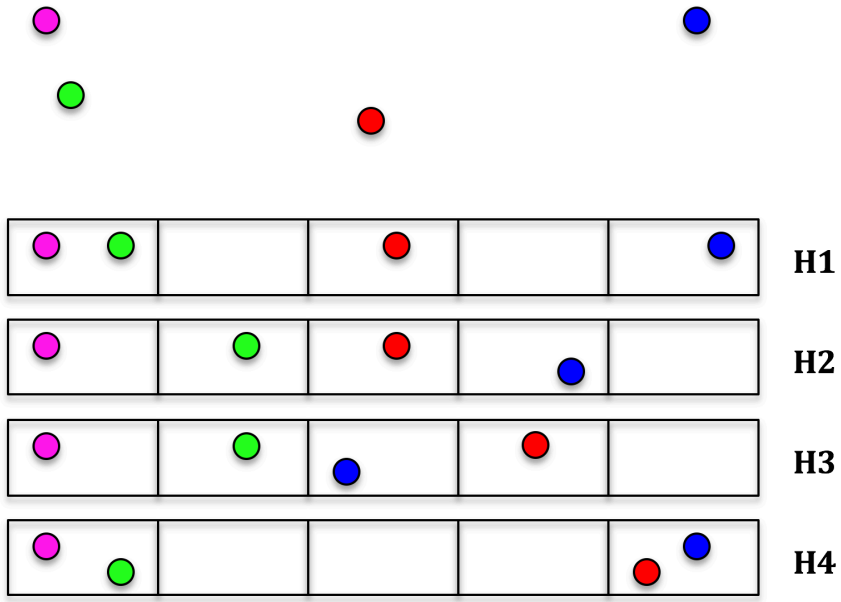


Locality Sensitive Hashing (LSH)

Data Preprocessing — To reduce the dimension of data

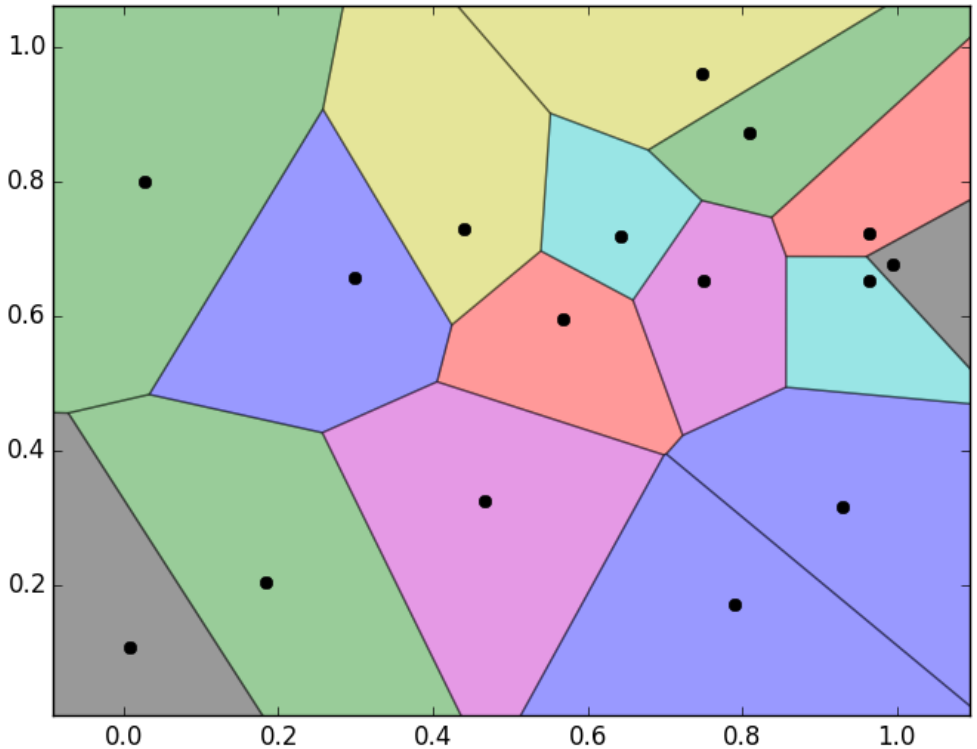
Space Filling Curve (Z-Value)

Locality Sensitive Hashing (LSH)



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

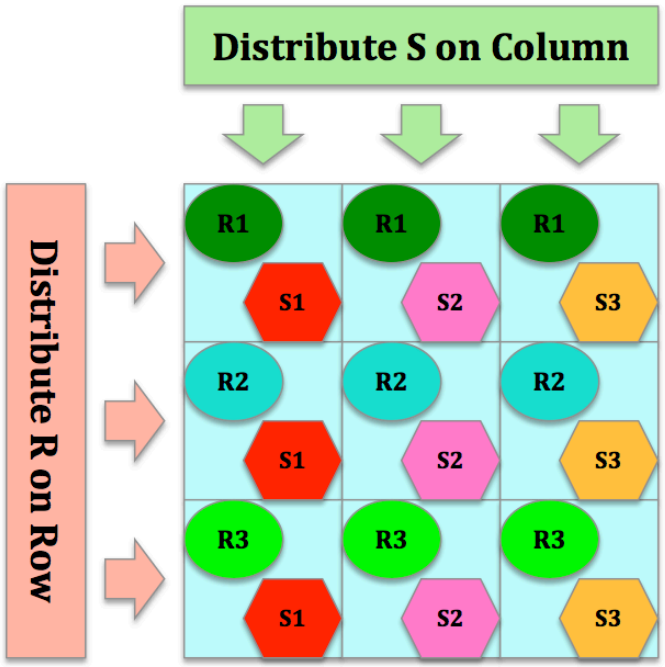
Voronoi Diagram:



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 rest 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 a cluster as the new pivot 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.

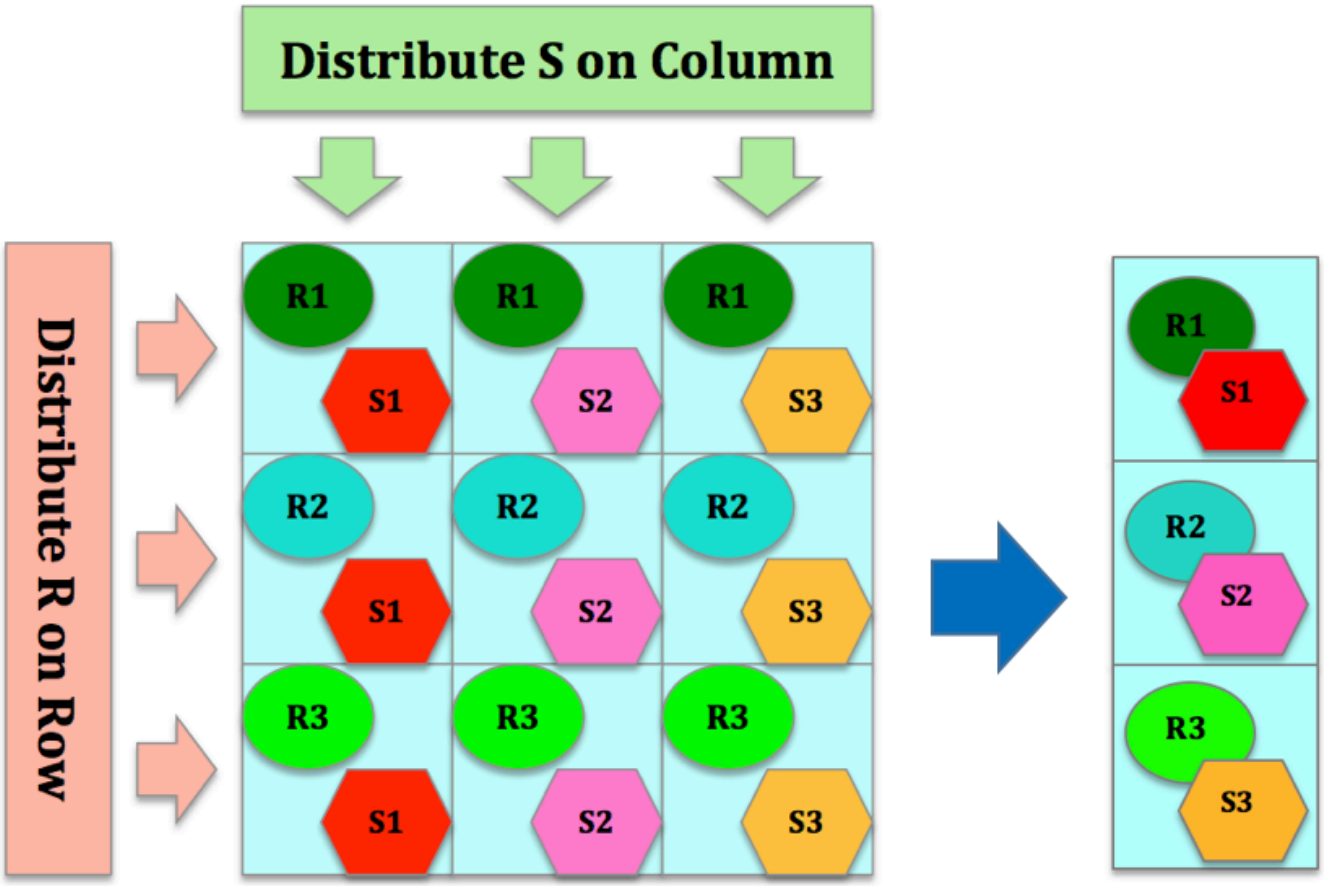
More precisely, what we want is: for every partition R_i ($\cup_i R_i = R$), find a corresponding partition S_j ($\cup_j S_j = S$), where:

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

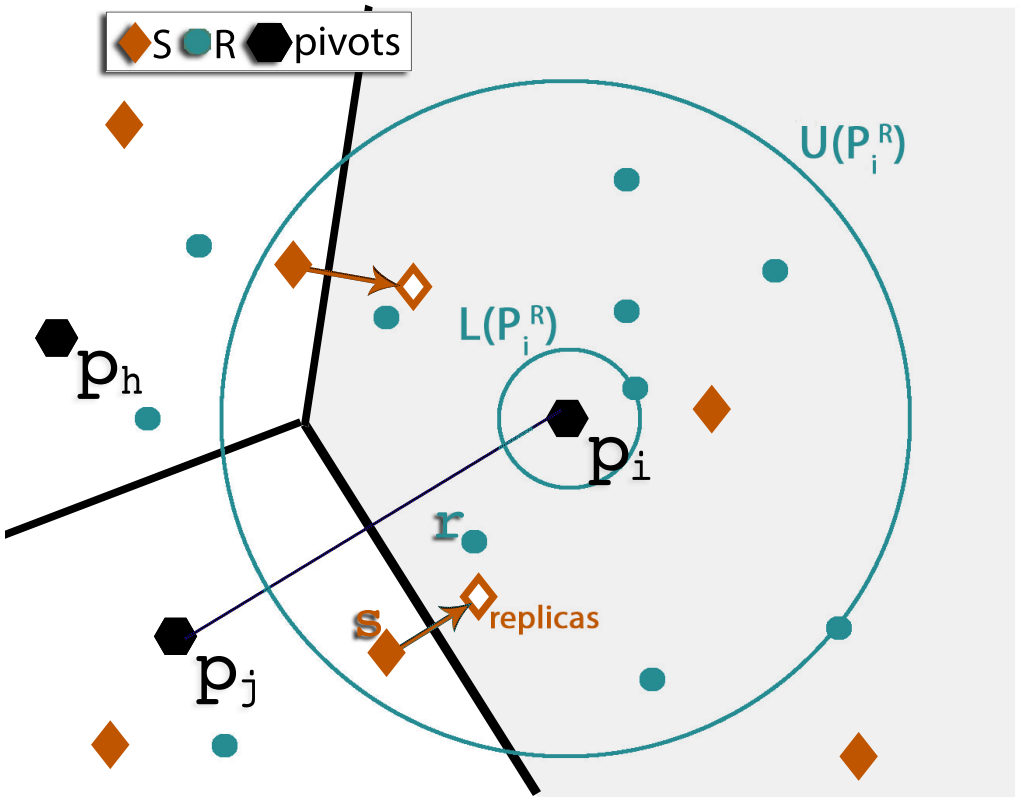
And,

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

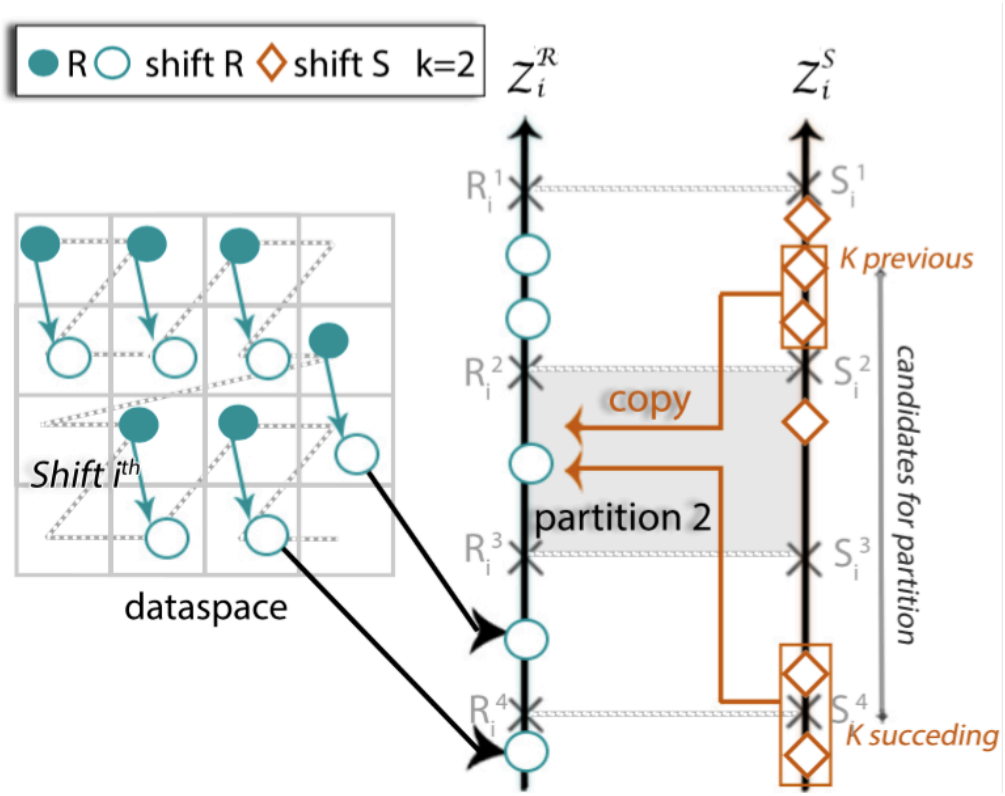
Data Partitioning – Motivation



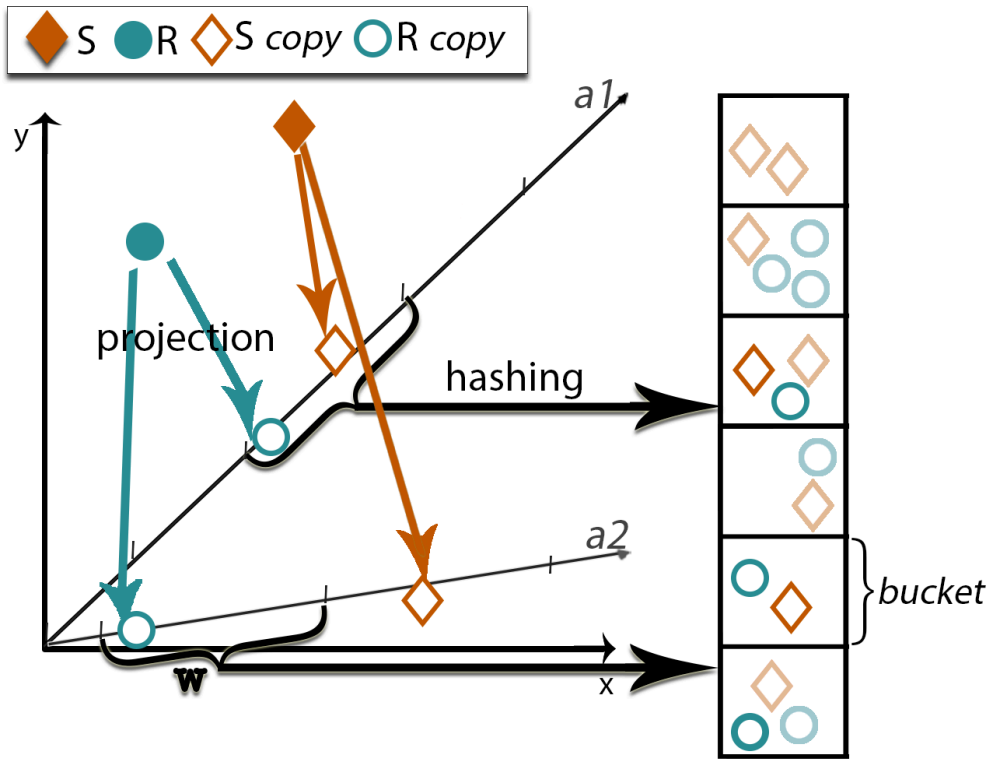
Data Partitioning — Distance Based Partitioning Strategy



Data Partitioning — Size Based Partitioning Strategy — Z-Value



Data Partitioning – Size Based Partitioning Strategy – LSH



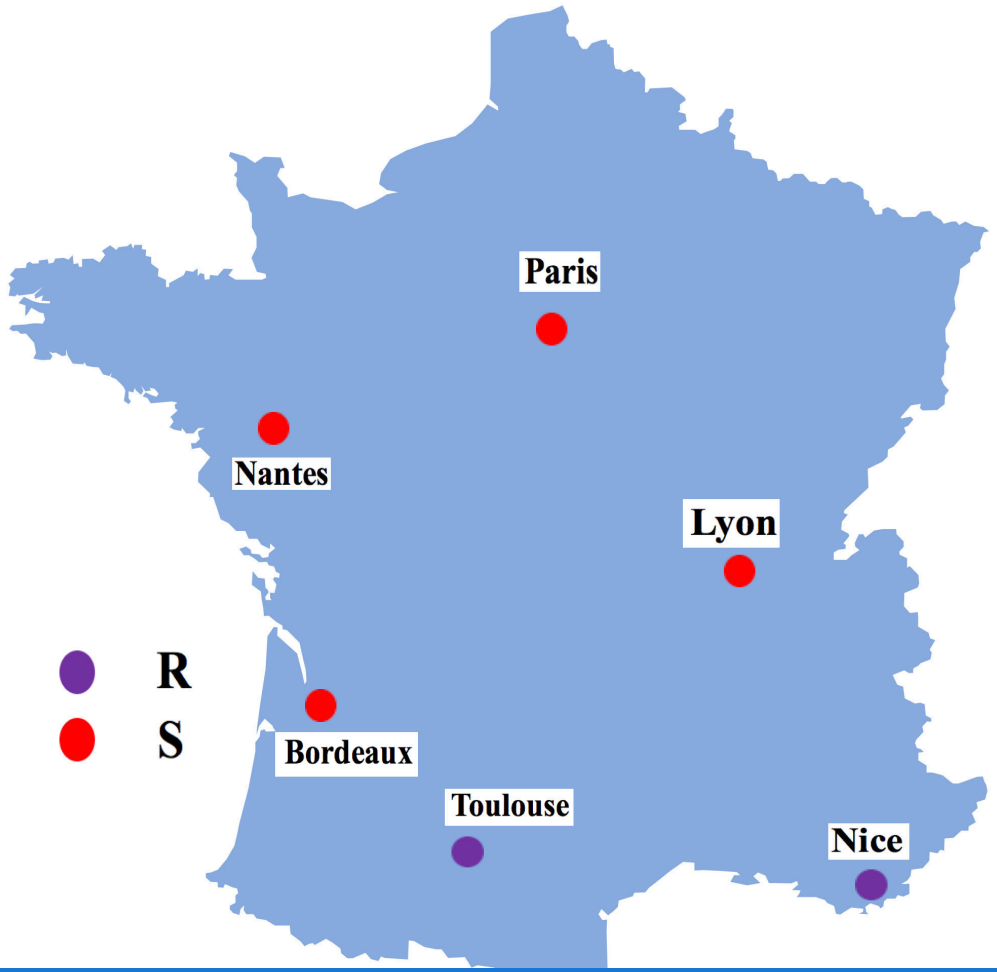
Computation

- 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.

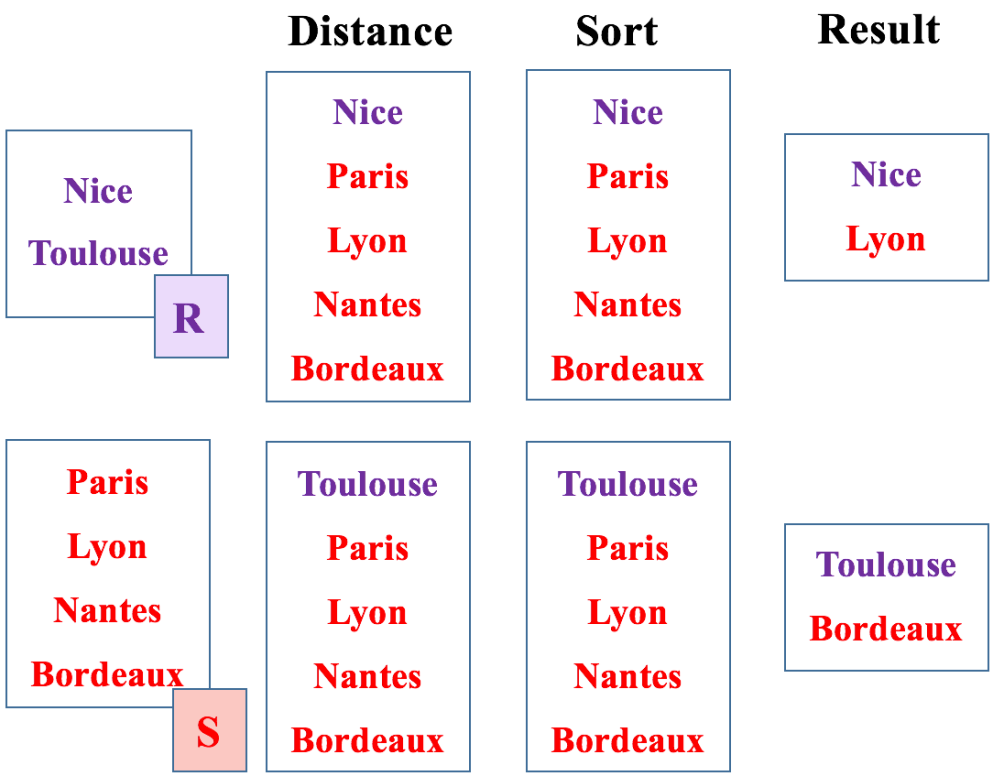
Computation – Example

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



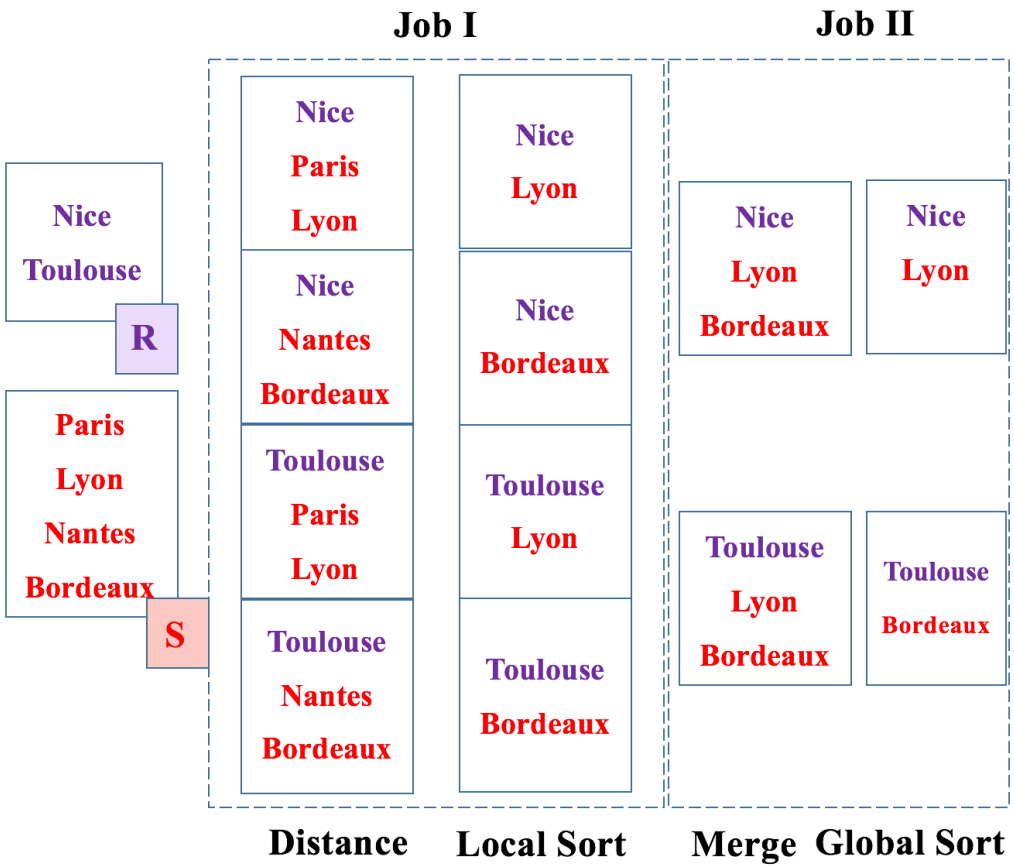
Computation – Example

One Job:

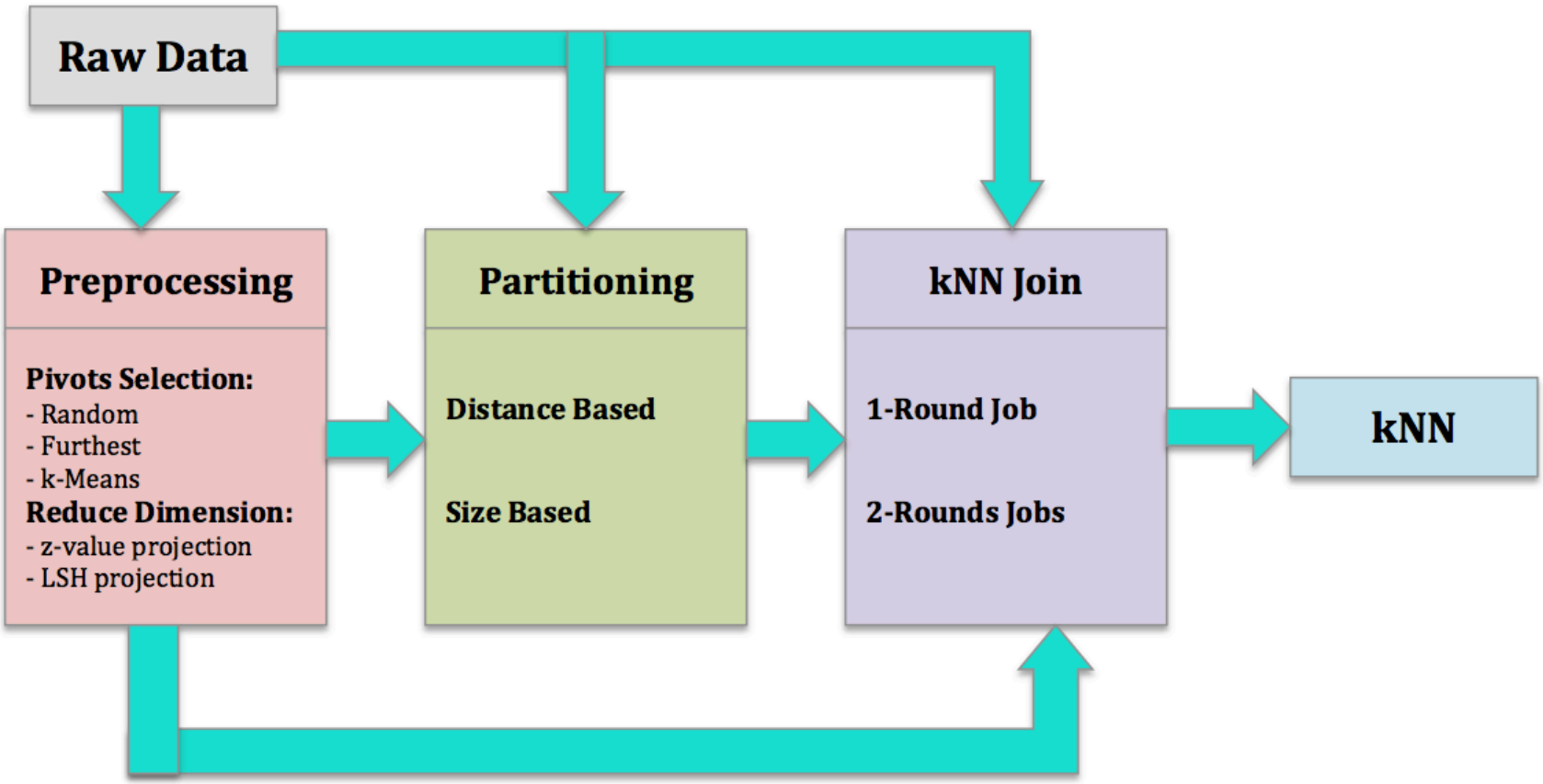


Computation – Example

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:

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

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

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

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

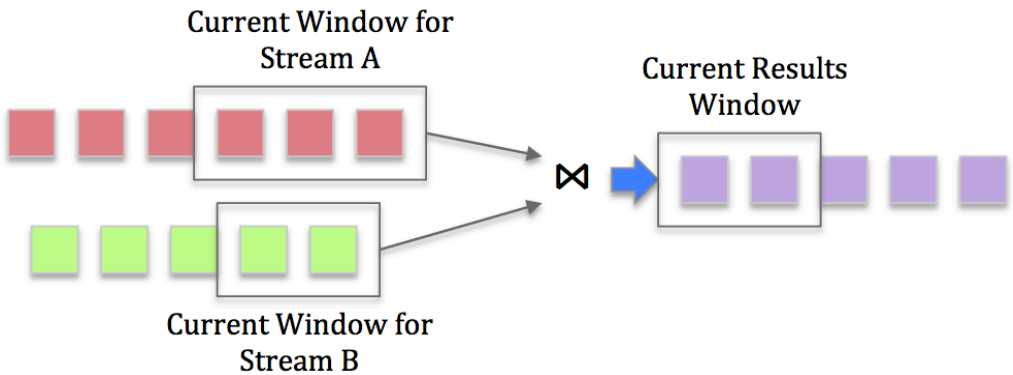
- Introduction
- Parallel Workflow
- Theoretical Analysis
- Continuous kNN
- Experiment Result
- Conclusion

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

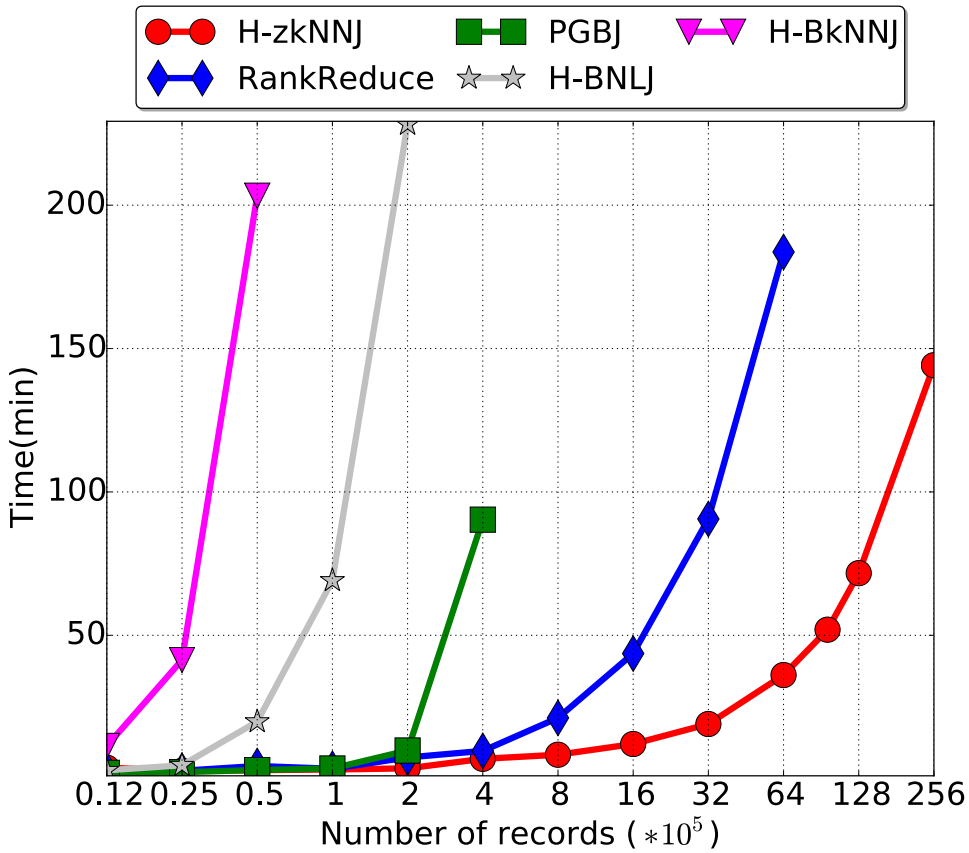
- **H-BkNNJ** Naive Method – Without preprocessing and partitioning – One Job
- **H-BNLJ** Block Nested Loop – Without preprocessing and partitioning – Two Jobs
- **PGBJ** Based on Voronoi – Preprocessing: Select Pivots – Distance Based Partitioning – One Job
- **H-zkNNJ** Based on Z-Value – Preprocessing: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

Evaluations

- Impact of input data size
- Impact of k
- Communication Overhead
- Impact of Dimension and Dataset
- Practical Analysis

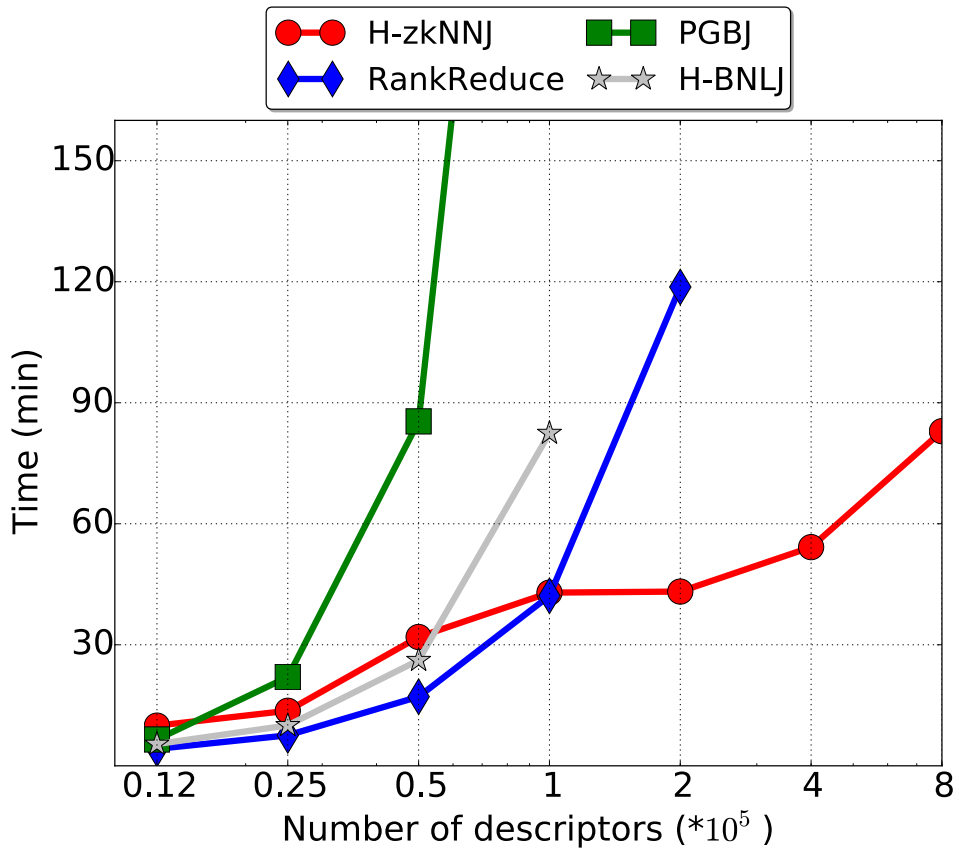
Evaluation Result — Verify the theoretical Analysis

Execution Time for Geo dataset (2 dimensions):



Evaluation Result — Surprise

Execution Time for Image dataset (128 dimensions):



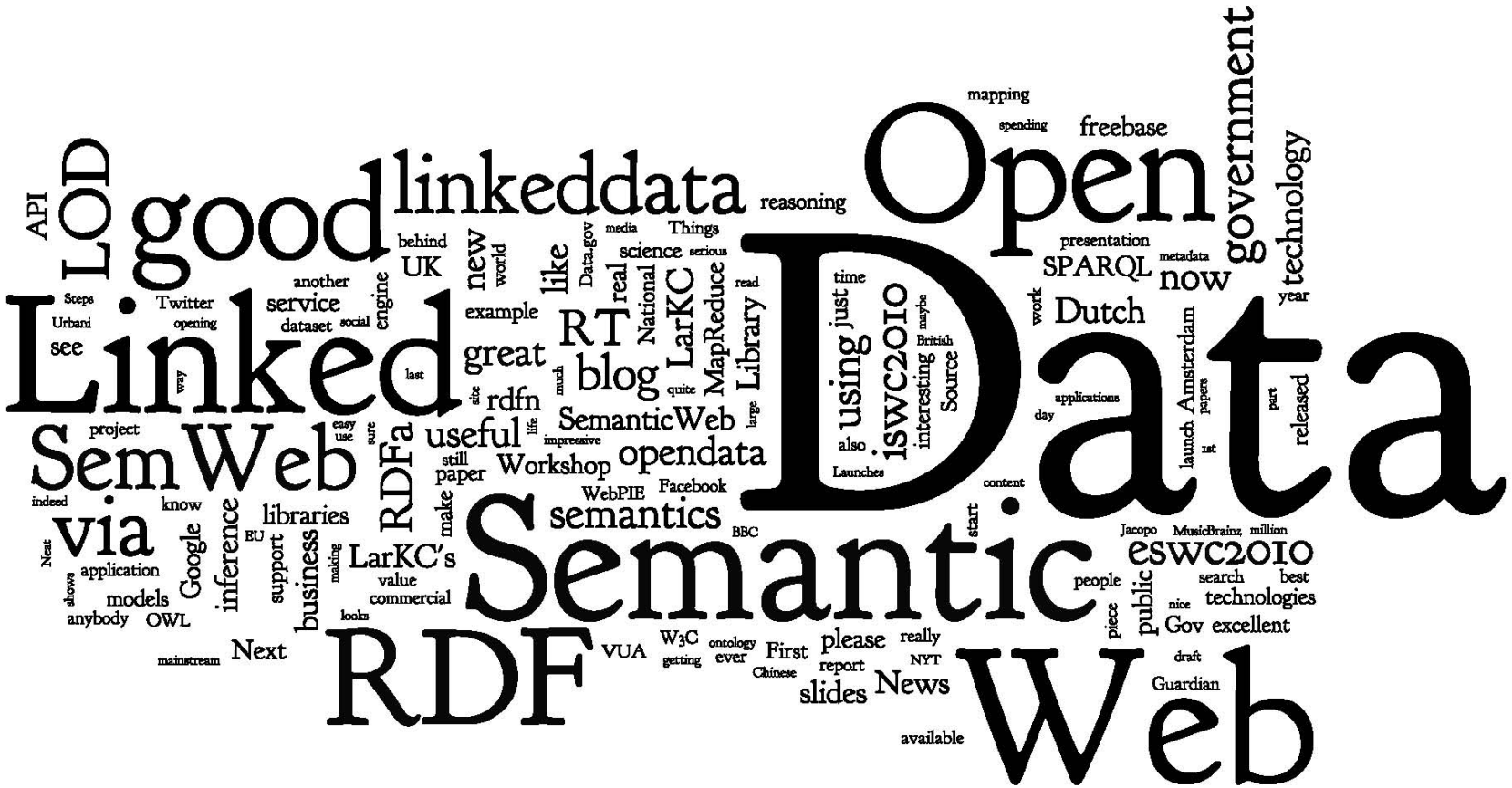
Outline

- 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

- Related Work
- Query Decomposition and Distribution
- Data Partition and Assignment
- Parallel and Distributed Query Plan
- Continuous Join
- Analysis
- Implementation
- Experiment Result
- Conclusion

Outline

- Related Work
- Query Decomposition and Distribution
- Data Partition and Assignment
- Parallel and Distributed Query Planner
- Continuous Join
- Analysis
- Implementation
- Experiment Result
- Conclusion

Thank You!