

---

# Gram Matrix Approximation Using Locality Sensitive Hashing on Cluster

---

Mohamed Hefeeda

Wael Abdel majeed

Fei Gao

Taher Dameh

Simon Fraser University



---

# Outline

- Background and the Problem
  - Gram matrix (G-matrix)
  - The problem we want to solve
- Approximated G-matrix Scheme
- How does Hadoop Work?
- Implementation and Evaluation
- Conclusion and Future Work

# Background

- **Example**

$$x_1 : d_{11}, d_{12}, d_{13}, d_{14}$$

$$x_2 : d_{21}, d_{22}, d_{23}, d_{24}$$

$$x_3 : d_{31}, d_{32}, d_{33}, d_{34}$$

...


$$x_n : d_{n1}, d_{n2}, d_{n3}, d_{n4}$$

## Gram Matrix

Kernel Function



$$G(x_1, x_2, x_3 \dots x_n) =$$

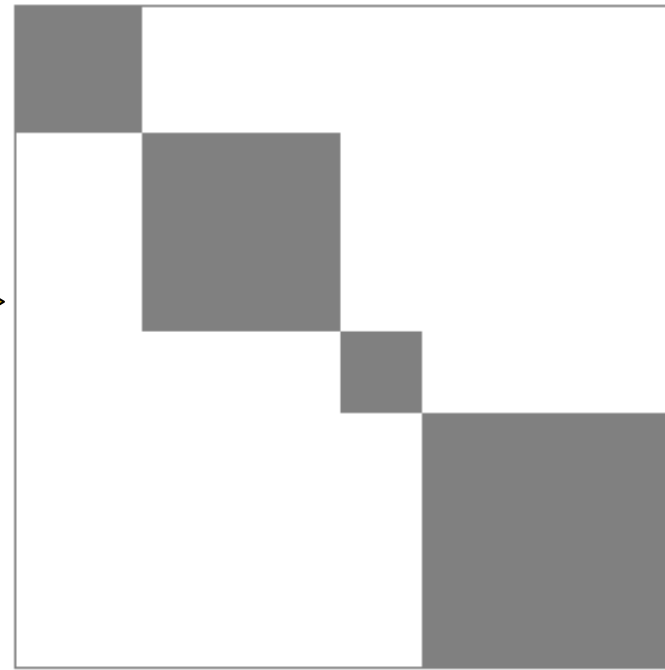
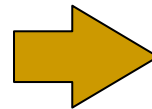

$$\begin{pmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \dots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \dots & \langle x_2, x_n \rangle \\ \vdots & \vdots & & \vdots \\ \langle x_n, x_1 \rangle & \langle x_n, x_2 \rangle & & \langle x_n, x_n \rangle \end{pmatrix}$$

# What if the data set is extremely large?

- Partition the dataset into different clusters based on similarity measurement
- Compute the sub-matrix in every cluster



Original Full G-matrix



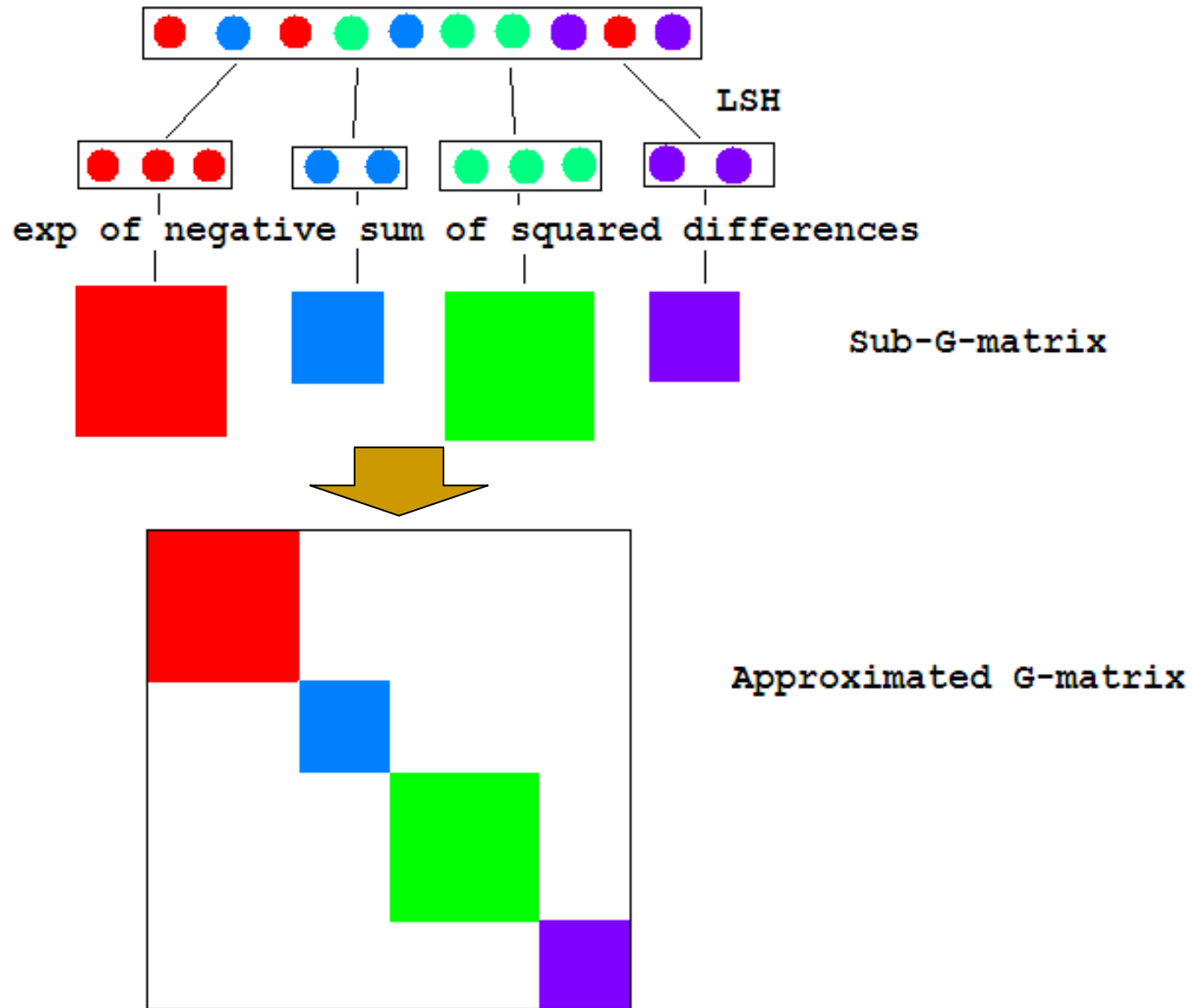
Approximated G-matrix

---

# Approximated G-matrix Scheme

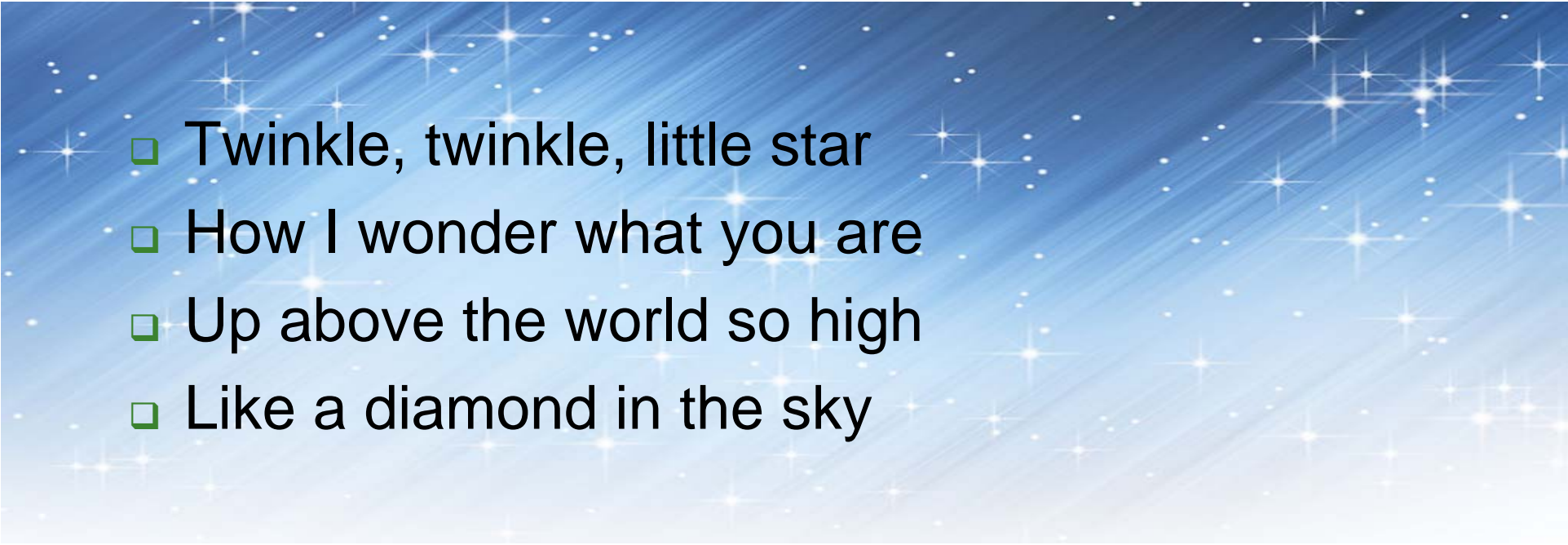
- Using locality preserving properties of **Locality Sensitive Hashing**<sub>[1]</sub> to approximate the original gram matrix
- **Parallelize** the program on cluster using Hadoop Map-reduce framework

# Approximated G-matrix Scheme (cont.)

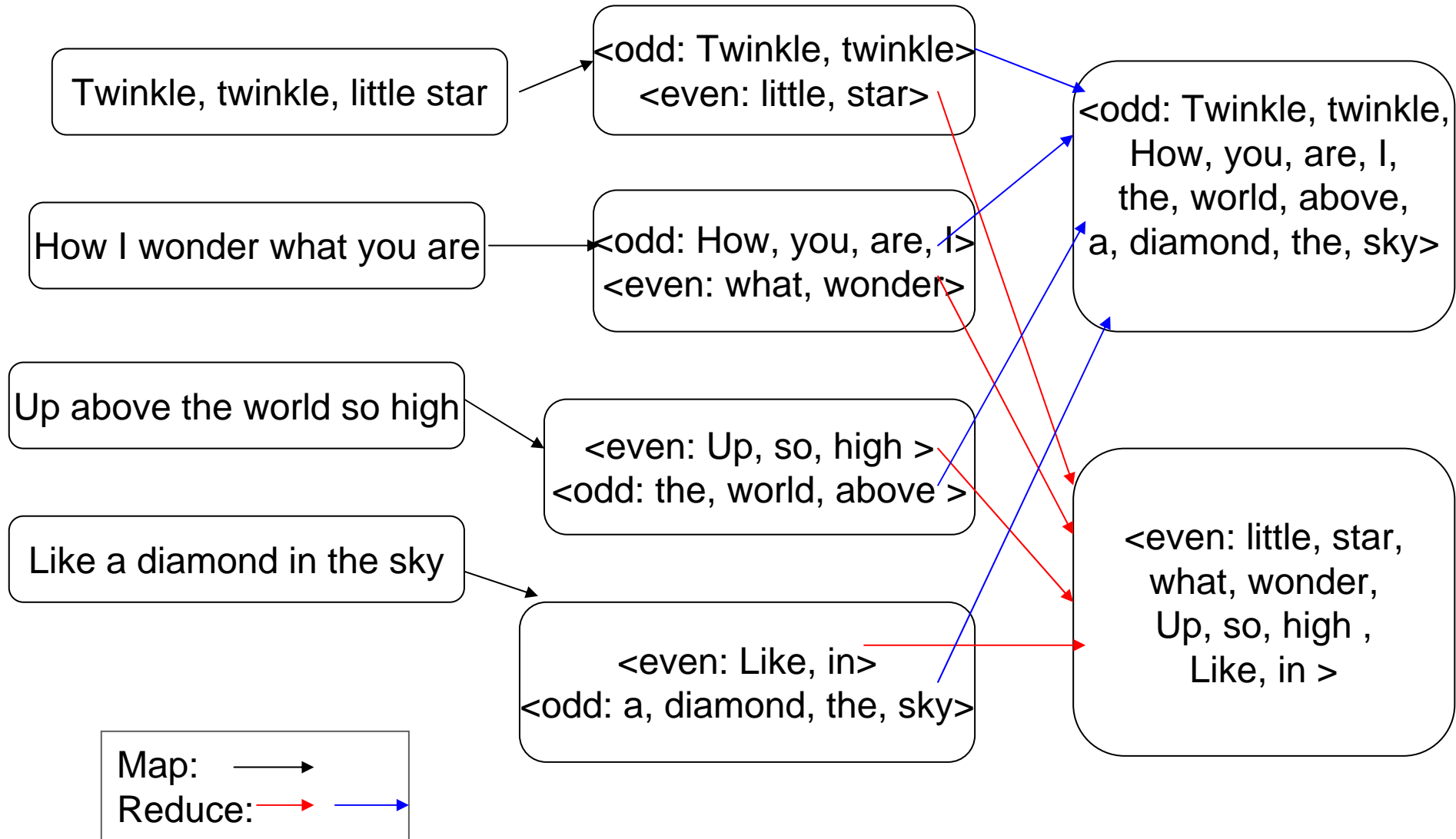


# How does Map-Reduce Work?

- **An example:** classify the words in the following nursery rhyme with respect to the parity of word length

- 
- ❑ Twinkle, twinkle, little star
  - ❑ How I wonder what you are
  - ❑ Up above the world so high
  - ❑ Like a diamond in the sky

# How does Map-Reduce Work?

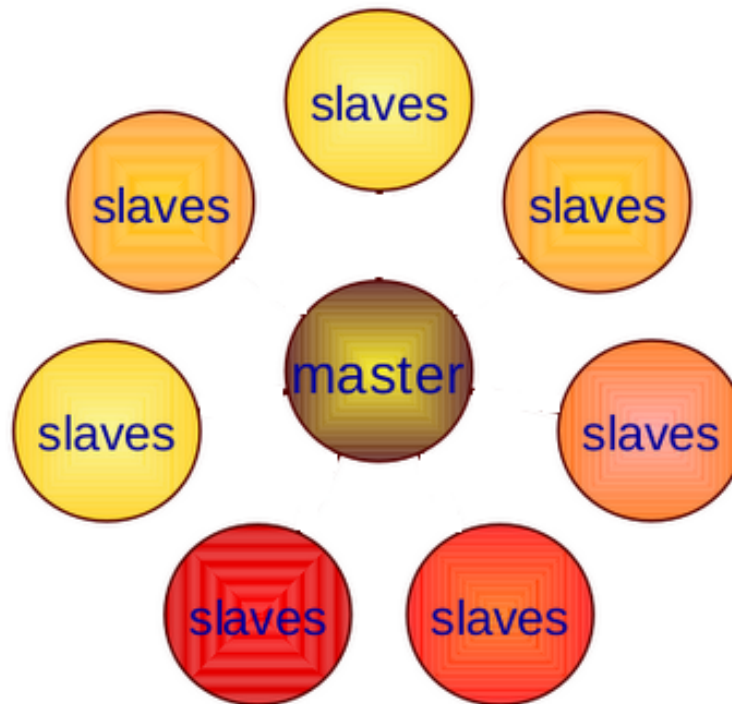




# Look into Hadoop



- Organization

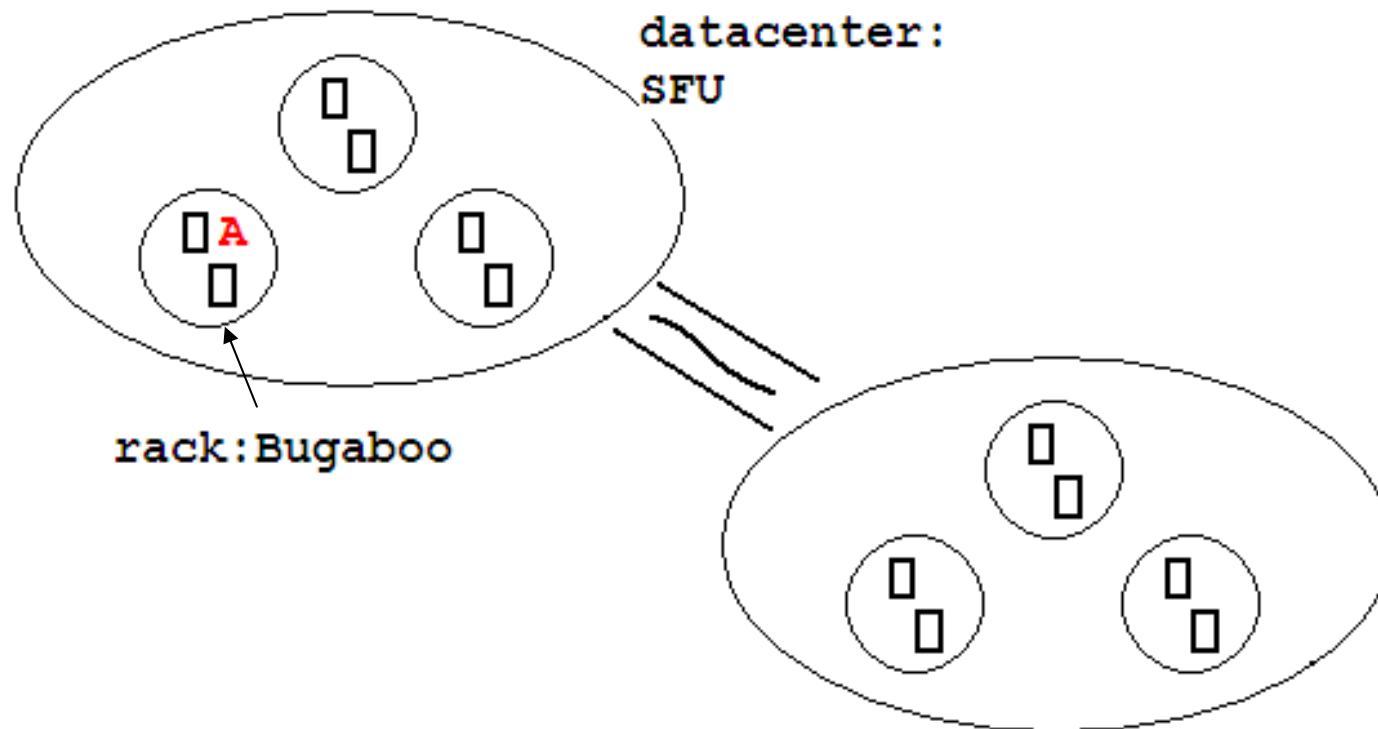


- Master is called *JobTracker*, slaves called *TaskTracker*

# Look into Hadoop (cont.)



- Rack, Data centre Determination



- The path of machine **A**: /SFU/Bugaboo/A

## Look into Hadoop (cont.)



- How Input Data is **Split** and **Distributed**?
  - ❑ Class *InputSplit* defines how to do the split of an input file.
  - ❑ In most cases Class *FileInputFormat* and its subclass will use 64MB (recommended) to split input file.
  - ❑ Function *computeReplicationWorkForBlock()* in Class *FSNamesystem* does the replication (default: 3 copies per block).
  - ❑ one copy on the local node, one copy on a node in the same rack, and the third copy on a node which is outside the rack.

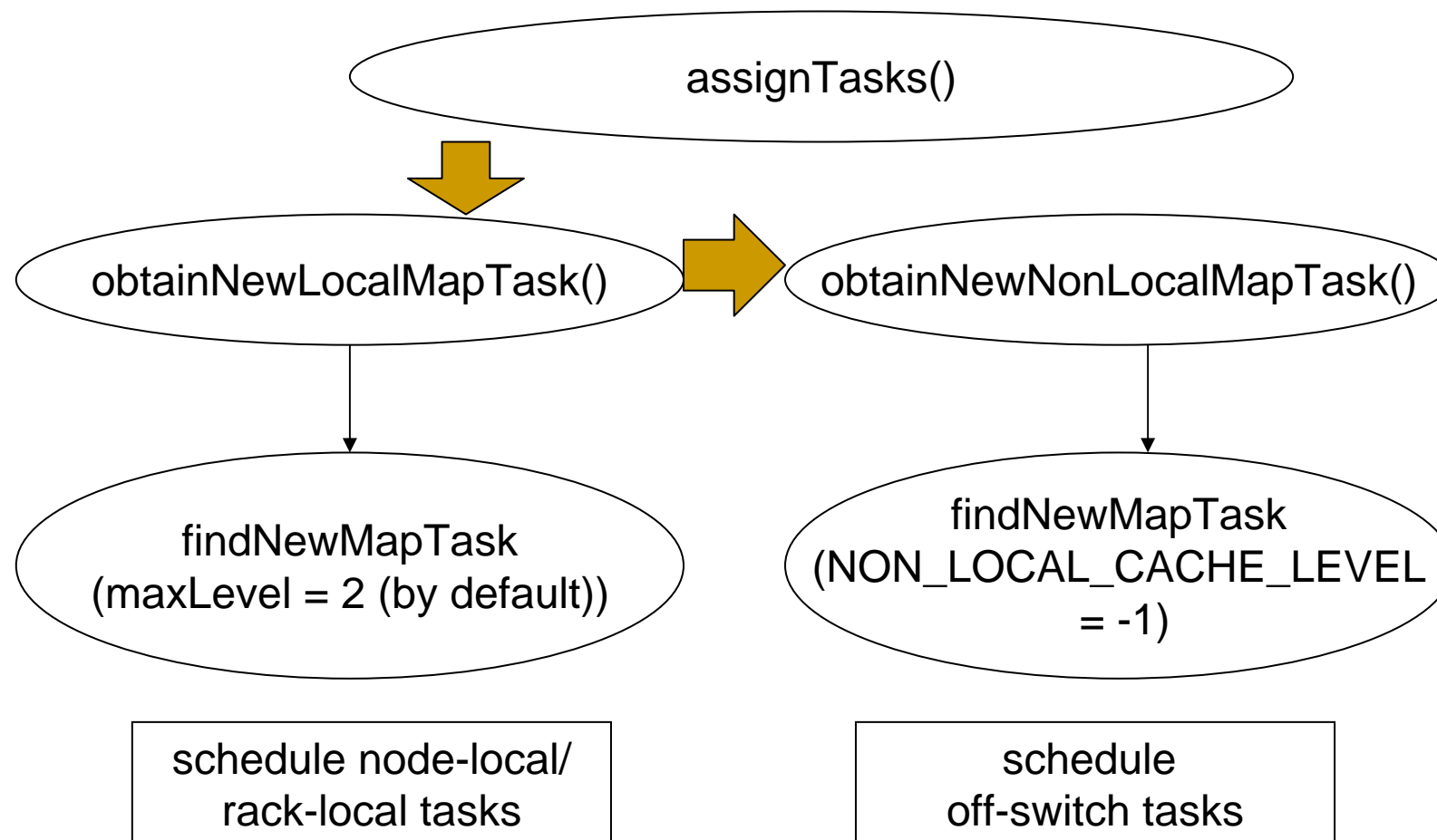
---

# Look into Hadoop (cont.)

- **Task Scheduling**

- ❑ Task assignment is initiated by a slave (asks for a task), then master decides which task to schedule and responses.
- ❑ The Job Tracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack, then machines in the same datacenter for the nearest located mapping tasks (data split).

# Look into Hadoop (cont.)



## Implementation Using Hadoop

- **Step 1: Mapping.** For every 64-dimension vector, do LSH in mapping part, the LSH method used is Random Projection from [www'07<sub>\[2\]</sub>](#). The output pair of this step is `<bucketNumber, vectorIndex>`.

$a[i] = \text{vector}[\text{dim\_}] \geq \text{threshold\_} ? 1 : 0$ , for every  $i \in [0, m-1]$

$\text{dim\_}$  and  $\text{threshold\_}$  are randomly generated

$$\text{BucketNumber} = \sum_{i=0}^{m-1} a[i] \cdot 2^i$$

Which is equal to concatenating these binary  $a[i]$  bits together. In this way, the maximum number of buckets is  $2^m$

# Implementation Using Hadoop

- **Step 2:** Hadoop **merges** the vectors which share the same bucket\_number together, into a list.
- **Step 3: Reducing.** For those vectors hashed into the same bucket, compute the similarity value between every two vectors to form the sub-Gram-matrix. Further, use clustering method.
  - Method of Similarity Computing

For 64-dimension vector  $a[]$  and  $b[]$ , and pre-set  $weight[]$

$$SimValue = e^{-\sum_{i=0}^{63} weight[i] \cdot (a[i] - b[i])^2}$$

---

## Experiment Setup

- **Single machine:** Intel® Xeon® Processor E5410 (12M Cache, 2.33 GHz, 1333 MHz FSB), 16 GB DRAM. GCC 4.1.3
- **Cluster:** six machines, each of which is Intel® Core™2 Duo Processor E6550 (4M Cache, 2.33 GHz, 1333 MHz FSB), 1 GB DRAM. Hadoop 0.20, JDK 1.6.0\_10
- **Dataset property:** 64 dimension, the numerical range of every component of the vector is  $[0,1]$



---

# Evaluation Methods

- **Hashing results** comparison between cluster and single machine
- **Frobenius norm (Fnorm)**
  - For m-by-n matrix,  $Fnorm = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$
- **Affinity Propagation** <sup>[3]</sup>
  - An unsupervised data clustering algorithm
  - Basic idea: given a set of data points with unknown cluster structure, the objective is to find a subset to serve as cluster exemplars.
- **Approximated G-matrix computing time**

---

# Hashing Results Comparison

- Procedure

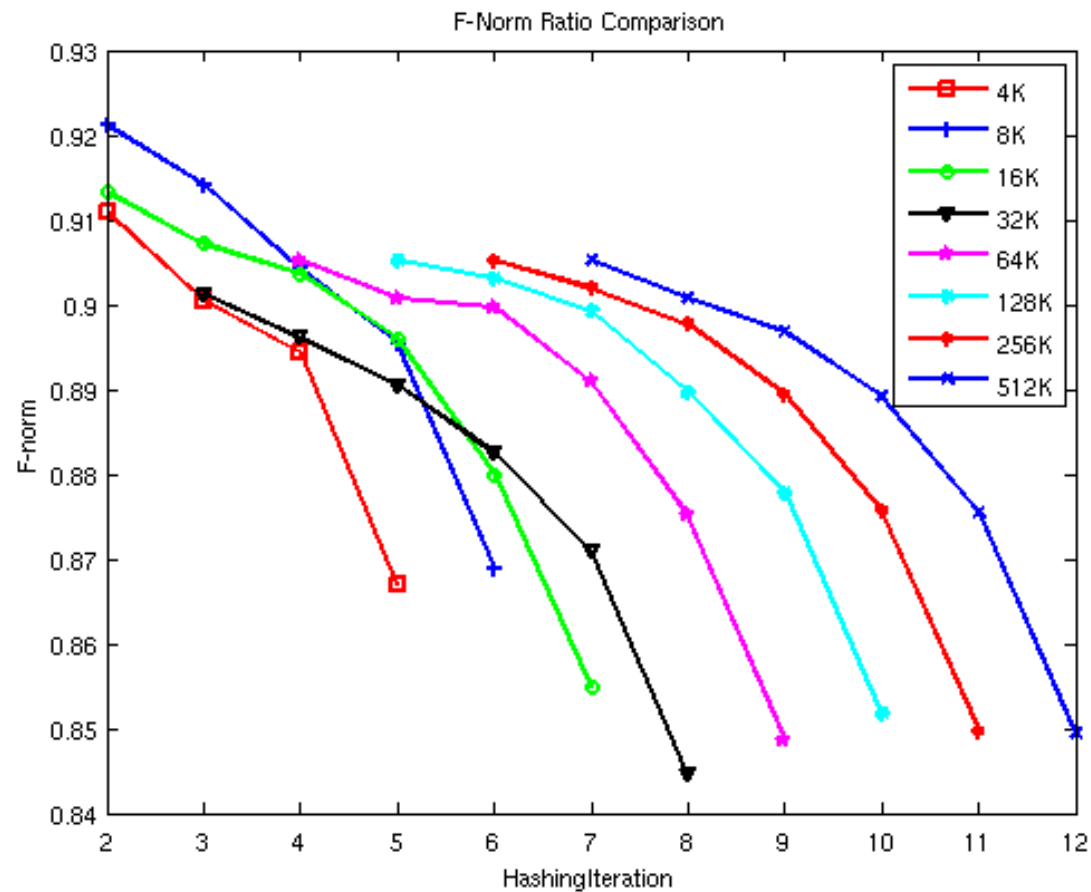
- Write the hashing results into files
- Use the same hashing parameter (hashing iteration 3, 6) and dataset (1K, 8K, 64K, 512K) to do hashing in cluster and single machine code.
- Sort the vector indices in every bucket (for efficiency)
- Count the number of same vectors ( $N_{same}$ ) in the corresponding bucket in cluster and single machine results.
- take the total of the two bucket size, and then subtract by  $2 * N_{same}$ , therefore, we get how many different vectors are in the same bucket

- Result

- The two implementations get exactly the same results, as expected.

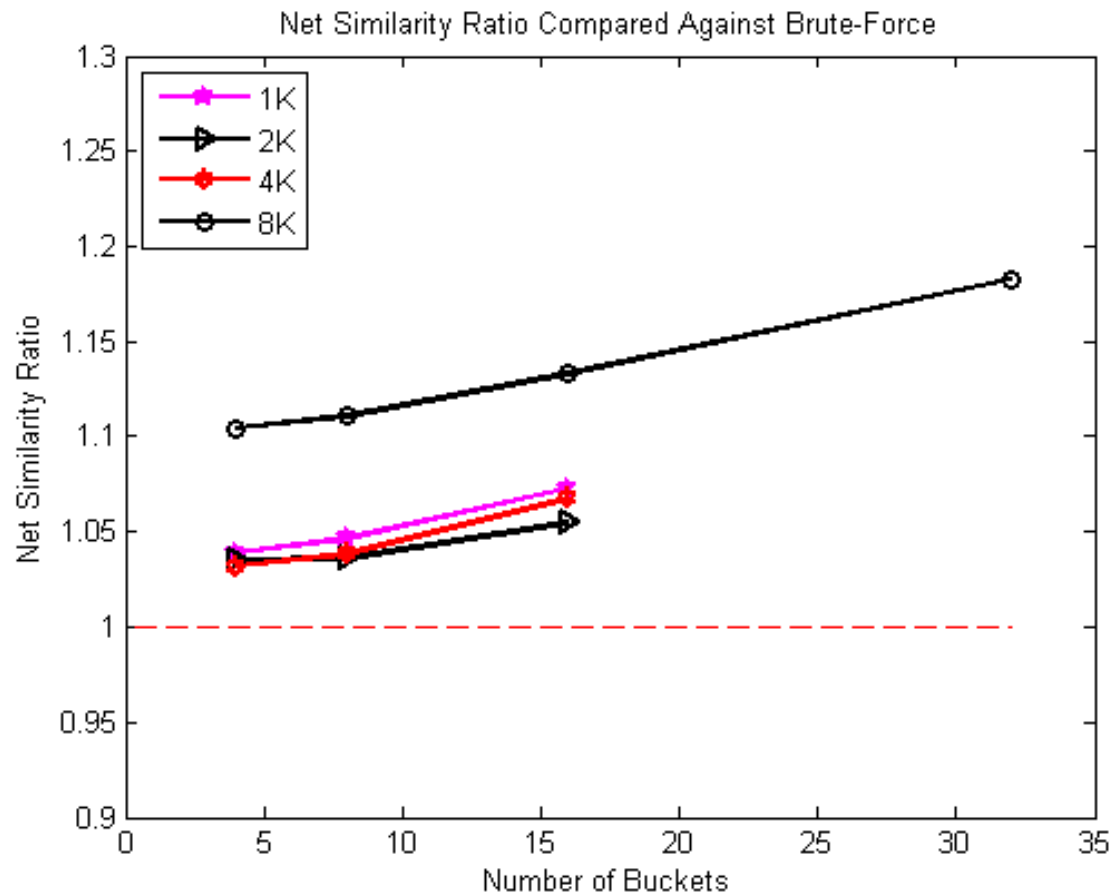
# Frobenius Norm Comparison

- Comparison between Approximated G-matrix and Original matrix



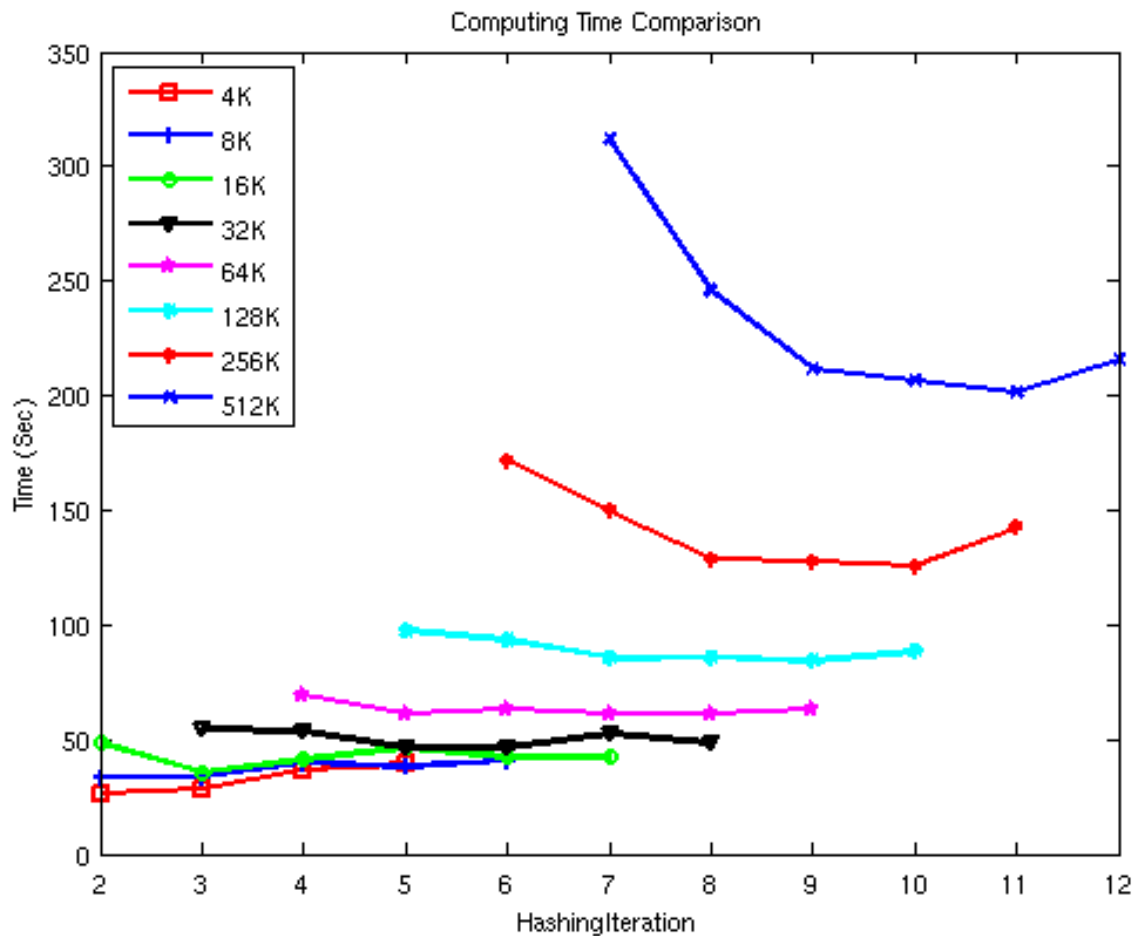
# Affinity Propagation Comparison

- The difference between Approximated and original matrix is small (5%-12% difference when 16 buckets)



# Approximated G-matrix computing time

- Time consumed grows sublinearly with the growth of data set size.



---

## Conclusion and Future Work

- High precision (6%-12% difference) can be achieved in Gram Matrix approximation using **Locality Sensitive Hashing (LSH)**
  - Frobenius norm (from matrix property side)
  - Clustering error (Net Similarity, from application side)
- **LSH** and **cluster** enables the parallelization of Gram matrix approximation and reduction of computing time
- Experiment on large scale cluster (Westgrid) to do optimization, further time reduction expected with guaranteed precision

---

# Reference

1. Gionis, A.; Indyk, P., Motwani, R. (1999). , "Similarity Search in High Dimensions via Hashing". Proceedings of the 25th Very Large Database (VLDB) Conference.
2. Gurmeet S. Manku, Arvind Jain, Anish D. Sarma: Detecting near-duplicates for web crawling, In WWW '07: Proceedings of the 16th international conference on World Wide Web (2007), pp. 141-150.

---

Thank you

Questions and Suggestions ( on  
deployment of cluster with 2304 cores)