MapReduce Project

Matrix-Multiplication

Hardik Shah
Kevin Shah
Ruturaj Nene

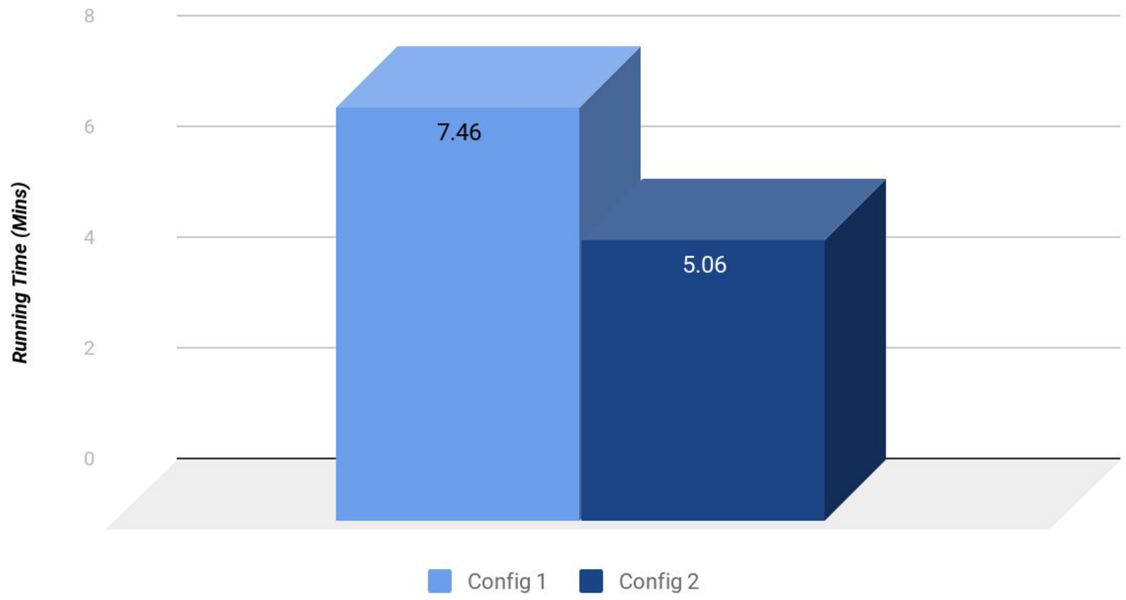
Overview

- Matrix Multiplication Sparse H-V
- PageRank Sparse H-V
- Matrix Multiplication Dense B-B
- Analysis and Experiments

Matrix Multiplication - Sparse H-V partitioning

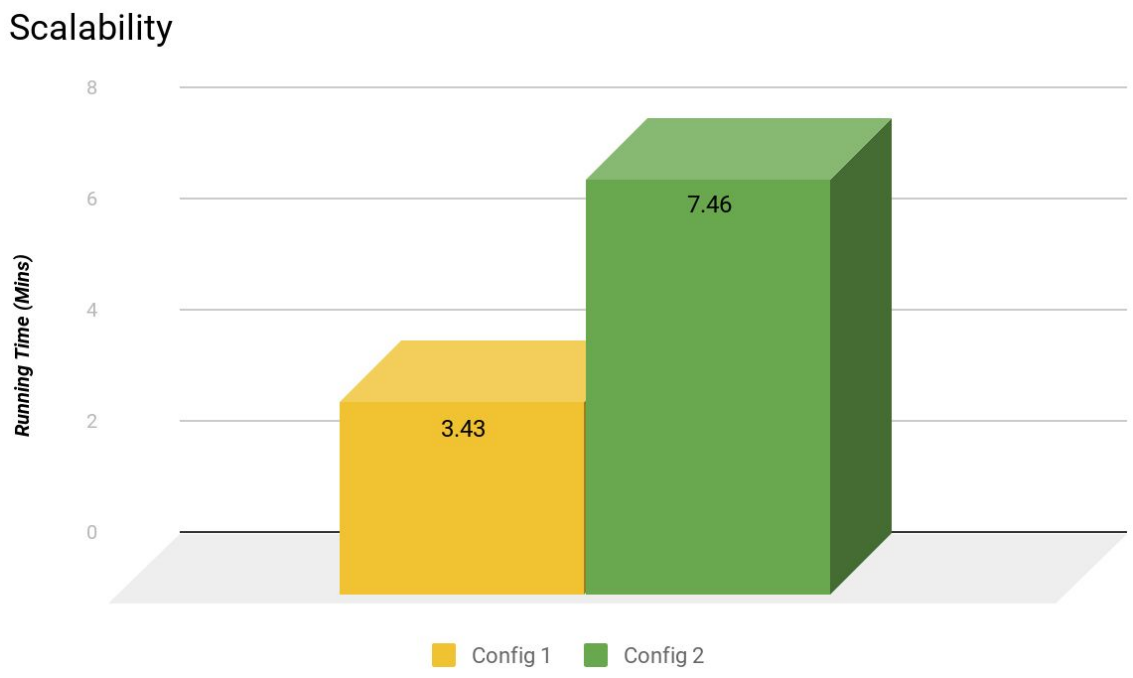
- Partition left matrix horizontally (H)
- No need to partition right matrix (Nx1 Vector)
- Duplicate right vector H times
- Each reducer receives group of rows from left matrix and whole right vector, multiplies each row with the column vector

Speedup



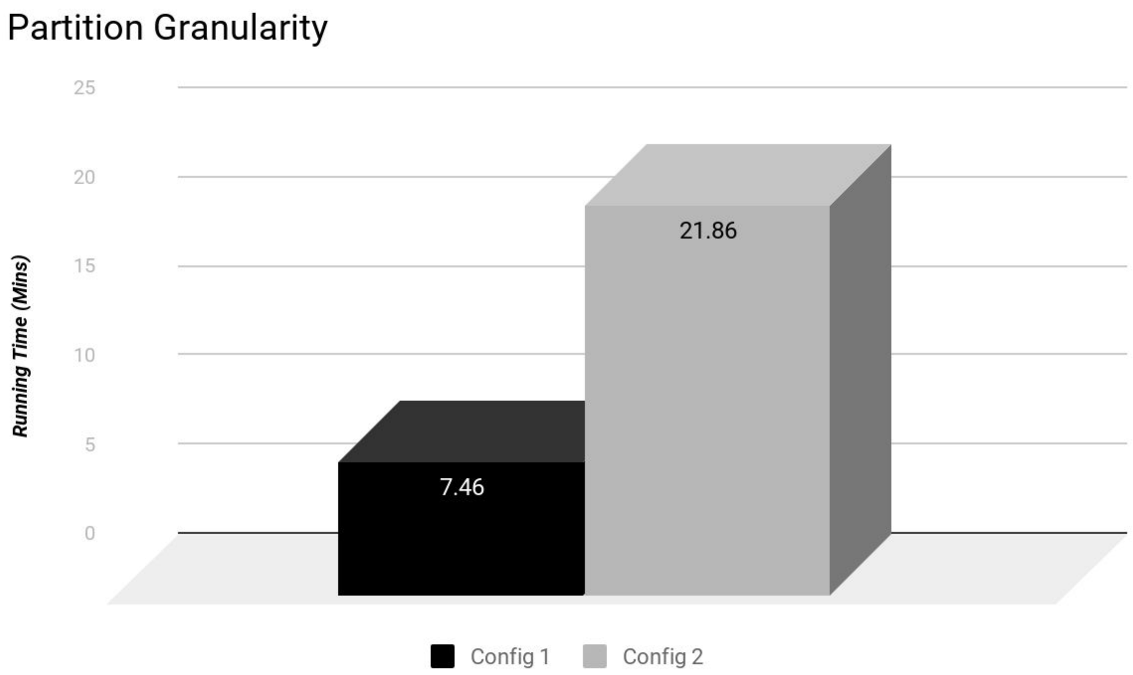
Config 1- Matrix size: 30k x 30k and 30k x 1, 4 workers, 20 Partitions(H)

Config 2- Matrix size: 30k x 30k and 30k x 1, 8 workers, 20 Partitions(H)



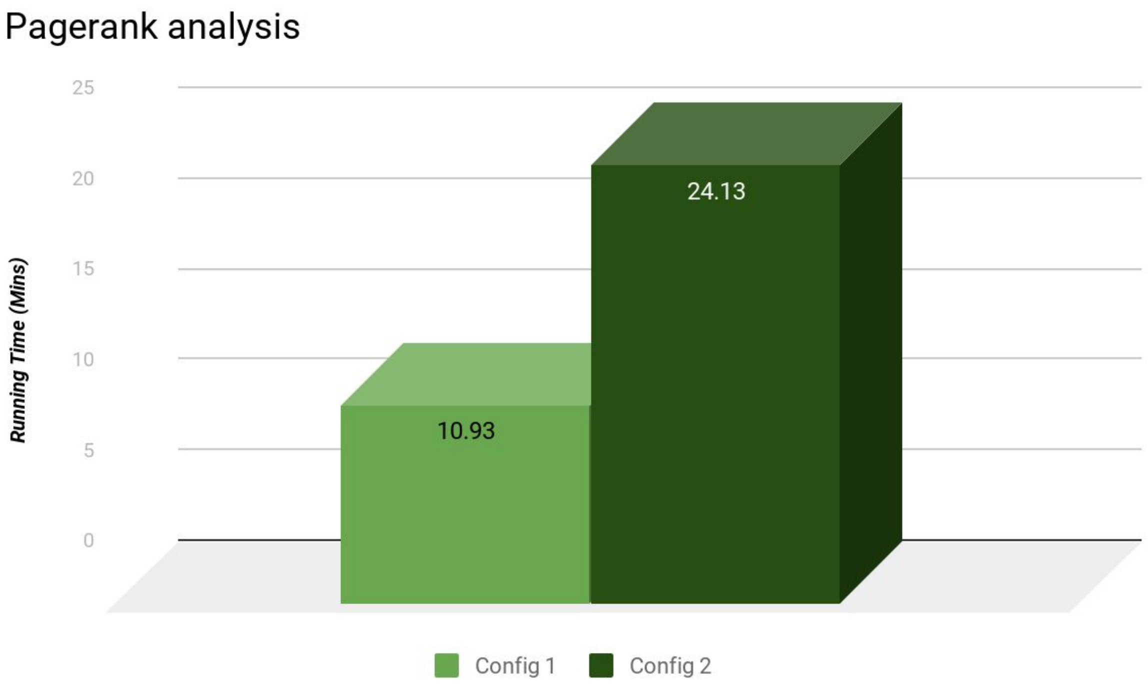
Config 1- Matrix size: 20k x 20k and 20k x 1, 4 workers, 20 Partitions(H)

Config 2- Matrix size: 30k x 30k and 30k x 1, 4 workers, 20 Partitions(H)



Config 1- Matrix size: 30k x 30k and 30k x 1, 4 workers, 20 Partitions(H)

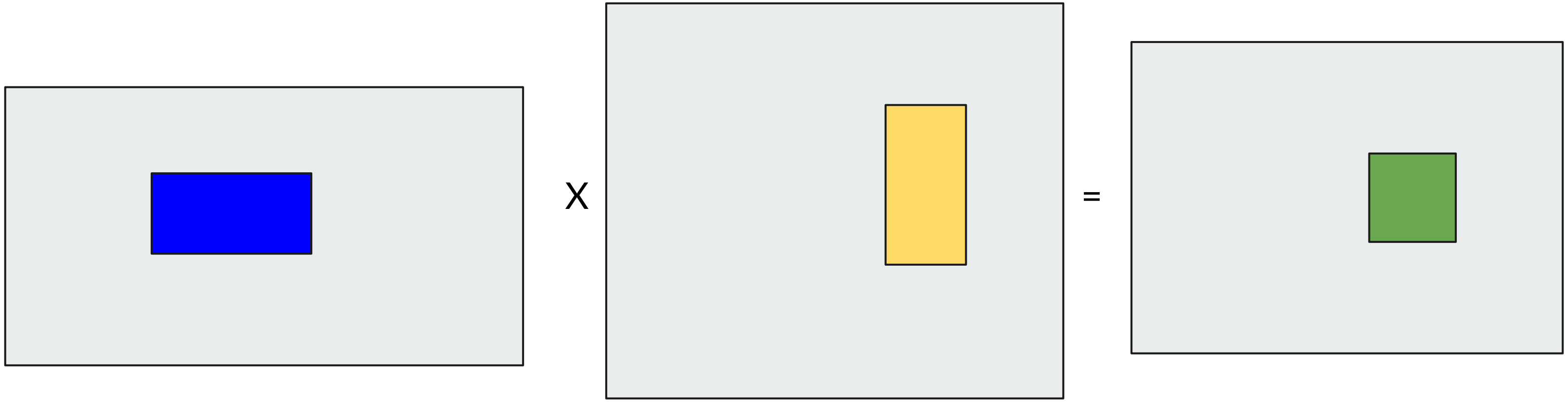
Config 2- Matrix size: 30k x 30k and 30k x 1, 4 workers, 20k Partitions(H)



Config 1- Synthetic graph from HW4, k=500, 10 workers

Config 2- Matrix size: 250k x 250k and 250k x 1, 10 workers, 25 Partitions(H)

Matrix Multiplication (Dense B-B partitioning)



Matrix A - Region (I, J)

Matrix B - Region (J, K)

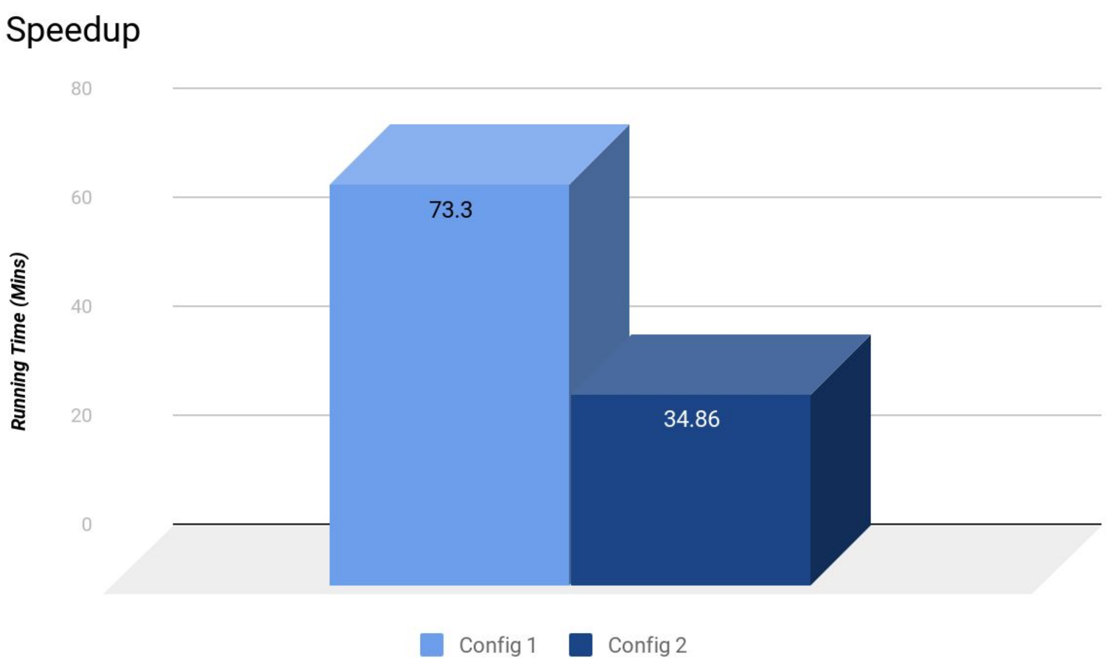
for i in I and k in K, contribution = $\sum A_{ij} * B_{jk}$ for j in J

Matrix Multiplication (Dense B-B partitioning)

- JOB 1: Divide left and right matrix into block partitions (H1xV1 , V1xV2) and computes partial sums of certain matrix products
  - Mappers:
    - A_{ij} -> key = (I,J,K) for all K Partitions, value = ("A", i, j, A_{ij})
    - B_{jk} -> key = (I,J,K) for all I Partitions, value = ("B", j, k, B_{jk}), where i,j,k are part of partitions I,J,K respectively
  - Reducers:
    - For key = (I, J, K) compute $x_{ik} = \sum A_{ij} * B_{jk}$ For all j in J, which is the partial sum for matrix product C_{ik} corresponding to a partition J

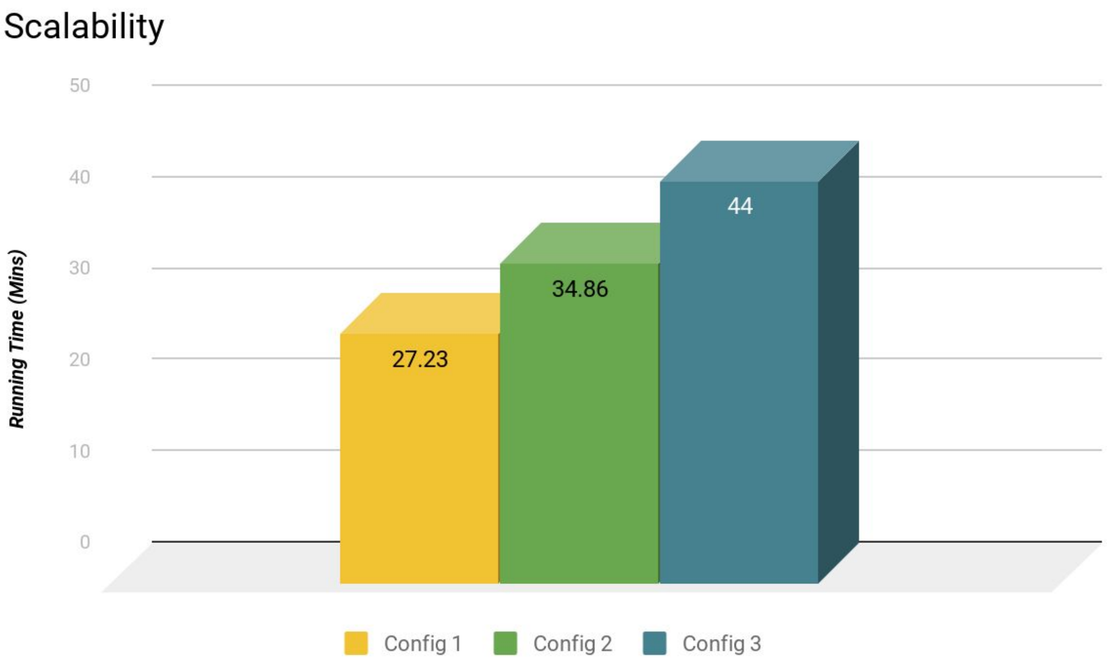
Matrix Multiplication (Dense B-B partitioning)

- JOB 2: Sum the partial sums and emit the matrix product
  - Mappers:
    - x_{iJk} -> key = (i), value =(k, x_{iJk})
  - Reducers:
    - For key = (i), accumulate partial sums for all k in a HashMap over all J Partitions
    - Emit output $C_{ik} = \sum x_{iJk}$ for all J Partitions, for all k in the HashMap



Config 1- Matrix size: 6k x 6k and 6k x 6k, 5 workers, H1, V1, V2 = 10

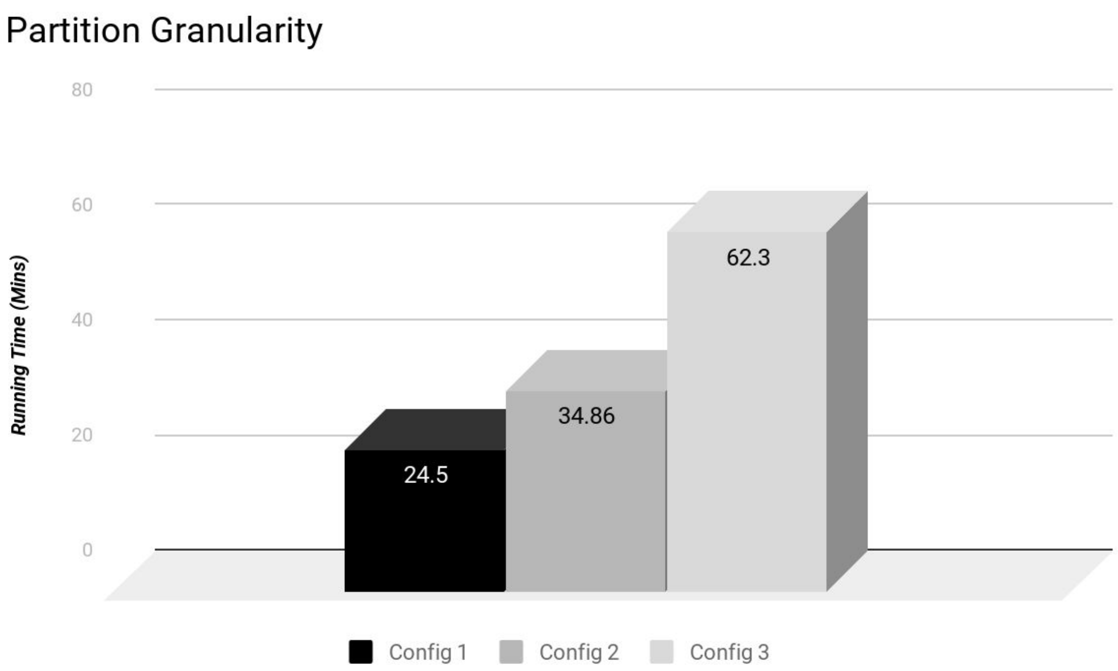
Config 2- Matrix size: 6k x 6k and 6k x 6k, 10 workers, 10, H1, V1, V2 = 10



Config 1- Matrix size: 5k x 5k and 5k x 5k, 10 workers, H1, V1, V2 = 10

Config 2- Matrix size: Matrix size: 6k x 6k and 6k x 6k, 10 workers, H1, V1, V2 = 10

Config 3- Matrix size: Matrix size: 7k x 7k and 7k x 7k, 10 workers, H1, V1, V2 = 10



Config 1- Matrix size: 6k x 6k and 6k x 6k, 10 workers, H1, V1, V2 = 5

Config 2- Matrix size: 6k x 6k and 6k x 6k, 10 workers, H1, V1, V2 = 10

Config 3- Matrix size: 6k x 6k and 6k x 6k, 10 workers, H1, V1, V2 = 20

Conclusion

- Achieved good scalability and speedup on Sparse H-V and Dense B-B matrix multiplication
- Proved concepts taught in class like increasing partition granularity can lead to worse performance because of higher duplication
- Dense B-B can be used for any application that requires matrix multiplication. Eg: Pagerank