Matrix Chain Multiplication via Multi-way Join Algorithms in MapReduce

Jaeseok Myung
School of Computer Science and Engineering
Seoul National University
jsmyung@europa.snu.ac.kr

Sang-goo Lee
School of Computer Science and Engineering
Seoul National University
sglee@europa.snu.ac.kr

ABSTRACT

In this paper, we translate the multiplication of several matrices into a multi-way join operation among several relations. Matrix multiplication is widely used for many graph algorithms, such as those that calculate the transitive closure. These algorithms benefit from the multi-way join operation because this operation reduces the number of binary multiplications. Our implementation is based on the MapReduce framework, allowing us to provide scalable computation for large matrices. Although several papers have investigated matrix multiplication using MapReduce, this paper takes a different perspective. First, we expand the problem from binary multiplication to n-ary multiplication. For this reason, we apply the concept of parallelism, not only to an individual operation but also to the entire equation. Second, we represent a matrix as a relation consisting of (row, col, val) records and translate a multiplication into a join operation in database systems. This facilitates the efficient storage of sparse matrices, which are very common in real-world graph data, and the easy manipulation of matrices. Although this work is still in progress, we conducted a number of experiments to verify the idea. We also discuss current limitations and future works.

Categories and Subject Descriptors

G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; D.1.3 [**Programming Techniques**]: Concurrent Programming; H.2.8 [**Database Management**]: Database Applications—*Scientific databases*

General Terms

Algorithms, Measurement, Performance

Keywords

Matrix multiplication, Multi-way join, MapReduce, Hadoop

1. INTRODUCTION

ICUIMC'12 February 20-22, 2012, Kuala Lumpur, Malaysia

Matrix multiplication is an important operation for many graph algorithms. For example, when finding n-hop neighbors from a social network, iterative multiplications of the zero-one adjacency matrix can produces the results by applying the transitive closure algorithm [7]. We can easily expand this idea to find connective components by computing transitive closures for all vertexes. Moreover, if we consider the weights of the edges, we can compute the distance between arbitrary two vertexes.

The semantics of matrix multiplication can be implemented by the join operation in database systems [5]. Suppose we have two n by n matrixes M_1 and M_2. The (i, j)-th element of $M_1 \times M_2$ is $\sum_{j=1}^{n} m1_{i,j} \times m2_{j,i}$. Now suppose we have two relations R_1 and R_2 whose attributes are $\{row, col, val\}$. A record in a relation corresponds to a non-zero element in a matrix. We can express the matrix multiplication in terms of SQL.

```
SELECT R1.row, R2.col, sum(R1.val*R2.val)
FROM R1, R2
WHERE R1.col=R2.row
GROUP BY R1.row, R2.col
```

The result of this query represents a matrix whose value of $(R_1.row, R_2.col)$-th element is $(R_1.val * R_2.val)$. Although representation schemes are different, the results of multiplication are equivalent. In this paper, we denote the matrix multiplication by $R_1 \bowtie_* R_2$.

The multiplication of several matrices also can be implemented using a join operation. Suppose we want to compute $M_1 \times M_2 \times M_3$. This can be solved through an iteration of binary multiplication, $(M_1 \times M_2) \times M_3$ or $M_1 \times (M_2 \times M_3)$. Another method of using the join operation is $(R_1 \bowtie_* R_2) \bowtie_* R_3$ or $R_1 \bowtie_* (R_2 \bowtie_* R_3)$. However, it should be noted that we can compute the multiplication of several matrices with a one-phase multi-way join operation in a database system. The multi-way join operation [4] combines multiple, separate two-way joins, meaning that we can compute $(R_1 \bowtie_* R_2 \bowtie_* R_3)$ at once. Therefore, we can easily deal with matrix chain multiplications or powers of a matrix using multi-way join algorithms.

MapReduce (MR) [8] is a good framework in which to implement a multi-way join operation for very large graphs and matrices. The size of graph data, for example, social networking data, is rapidly increasing. Given this trend, MapReduce has been employed as a scalable data processing framework by many researchers [10], [9]. The multi-way join operation can be easily implemented because the map function in the framework can divide records from different

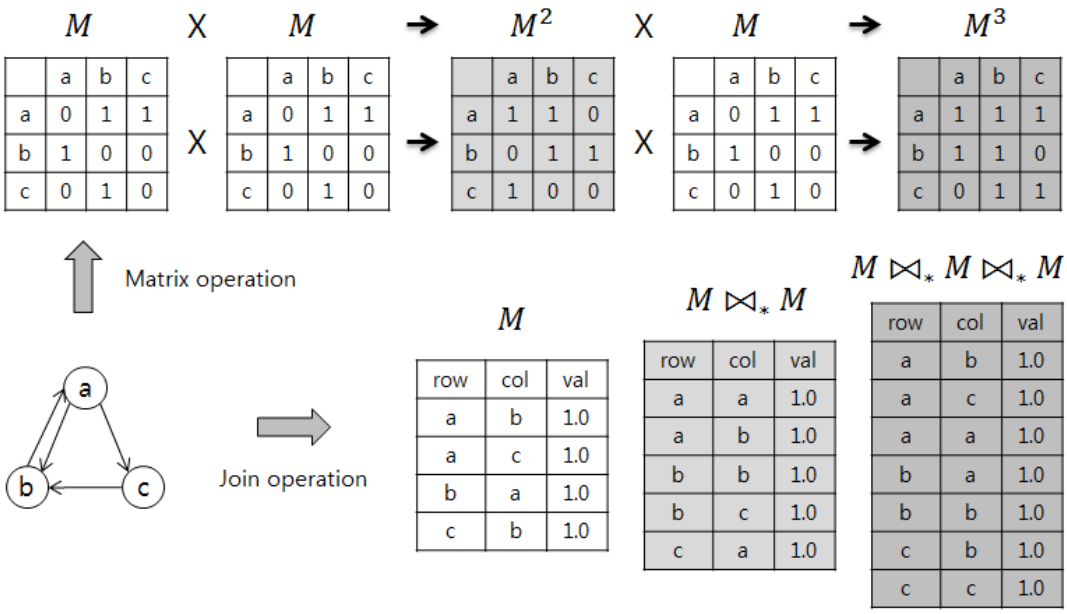


Figure 1: The semantics of matrix multiplication can be implemented by the join operation in a database system.

relations according to the join key. Therfore, the reduce function can join several records sharing the same join key at once. However, it is difficult to optimize the multi-way join operation in MapReduce due to IO and communication overheads despite the research done in this area [4], [11]. In this paper, we simply apply existing multi-way join algorithms to the matrix chain multiplication problem.

The contributions of this paper are as follows:

- We translate the matrix multiplication problem into a join operation in database systems. This viewpoint is important because researchers have an opportunity to optimize matrix multiplications. For example, we only represent a non-zero element of a matrix into a record consisting of (row, col, val), which make it unnecessary to consider the sparseness of a matrix. Another advantage of our approach is its flexibility. We can apply not only the multiplication but also addition, subtraction, and even custom functions such as max, min. By changing the value part of $R_1 \bowtie_* R_2$, we can easily define $R_1 \bowtie_+ R_2$, $R_1 \bowtie_- R_2$, and $R_1 \bowtie_{max} R_2$.
- We expand traditional MR-based binary matrix operations into operations for several matrices. Binary operations yield several MR jobs when we compute a matrix chain multiplication. A MR job takes considerable time for initialization. Therefore, minimizing the number of MR job iterations is important when optimizing the performance of MR-based algorithms. We first consider a serial two-way join operation which cascades the binary multiplication. After that, we utilze a parallel two-way join operation which computes several two-way joins simultaneously. Finally, we propose the m-way join approach for matrix chain multiplication.
- We verify the idea and report results from experiments on real graph datasets using Hadoop. With the results, we discuss the limitations of current research and suggest future works.

The rest of the paper is organized as follows. In Section 2, we introduce implementations of multi-way join algorithms for matrix chain multiplication. In Section 3, we show experimental results and discuss the pros and cons of the approach. Finally, we conclude the paper and suggest future works in Section 4.

2. ALGORITHMS

Matrix multiplication is associative. We already introduced the $\bowtie_*$ operation which is also associative. In other words, the following multiplications are all equivalent when we have matrices A, B, C, and D.

$$\begin{aligned} A \bowtie_* B \bowtie_* C \bowtie_* D &= (((A \bowtie_* B) \bowtie_* C) \bowtie_* D) \quad (1) \\ &= ((A \bowtie_* B) \bowtie_* (C \bowtie_* D)) \quad (2) \\ &= (A \bowtie_* B \bowtie_* C \bowtie_* D) \quad (3) \end{aligned}$$

There are a number of ways to compute the matrix chain multiplication. A typical method is to use sequential multiplication, as described in equation (1). We refer to this way as sa erial two-way join (S2). In S2, each $\bowtie_*$ operation requires a separate MR job. Although S2 employs the concept of parallelism, it is limited in its operation; i.e., it involves intra-operation parallelism. Another way to solve the above equation is through a parallel two-way join operation (P2). This approach follows inter-operation parallelism, which means we can compute $A \bowtie_* B$ and $C \bowtie_* D$ simultaneously. We can expand the concept of inter-operation parallelism so as to compute the matrix chain with a MR job, as represented in equation (3). We refer to this approach as a parallel m-way join operation (PM). In this section, we explain our implementation of these operations with the MapReduce framework.

Algorithm 1 driver function for serial two-way join

Input: Relations $M_1, M_2, ..., M_n$ representing matrices

1: $M_{left} = M_1$
2: **for** $i = 2$ to n **do**
3: $M_{right} = M_i$
4: M_{result} = doMR-S2 (M_{left}, M_{right})
5: $M_{left} = M_{result}$
6: **end for**

Algorithm 2 driver function for parallel m-way join

Input: Relations $M_1, M_2, ..., M_n$ representing matrices

1: $LIST\_M_{next} \Leftarrow M_1, M_2, ..., M_n$
2: **while** $|LIST\_M_{next}| > 1$ **do**
3: **for** $i = 1$ to $|LIST\_M_{next}|$ **do**
4: **if** $(i \ mod \ m) == 1$ **then**
5: add M_i to $LIST\_M_{left}$
6: $M_{left} = M_i$
7: **else**
8: add M_i to $LIST\_M_{right}(M_{left})$
9: **end if**
10: **end for**
11: $LIST\_M_{next}$ = doMR-PM $(LIST\_M_{left}, LIST\_M_{right})$
12: **end while**

2.1 Serial Two-way Join (S2)

In S2, we implemented the **repartition join** introduced in [6]. Because we are focusing on the matrix chain multiplication process, the *driver* function, which control the sequence of MR jobs, is important. In this section, we explain the *driver* function; the pseudo codes of the *map* and the *reduce* functions are outlined in the literature [6]. As described in Alg. 1, S2 computes the join operation between the first two relations. It then produces the next join results between the previous result relation and the next relation. A join operation needs a MR job.

2.2 Parallel M-way Join (P2, PM)

Several studies have examined binary multiplication between two matrices, such as [12], [9], and [13]. They all used the MapReduce framework in order to achieve the intra-operation parallelism. However, we focus on inter-operation parallelism in this study. Therefore, the *driver* function of PM should be implemented in a different way.

Alg. 2 shows a pseudo code of the *driver* function for a parallel m-way join. It should be noted that the two-way join is a special case of the m-way join, where m is 2. In the algorithm, we prepare a list of matrices for a MR job. We refer to this list as $LIST\_M_{next}$. Every matrix will be included on the list for the first job. We then add every $(m+1)$-th matrix to another list, $LIST\_M_{left}$. Every matrix in $LIST\_M_{left}$ has a list of matrices that will be multiplied. We denote these individual lists for M_{left} as $LIST\_M_{right}(M_{left})$. As a result, a MR job can compute all multiplications for each left matrix in the list $LIST\_M_{left}$. The result will be passed to the next MR iteration when the number of matrices in the $LIST\_M_{next}$ is greater than 1.

The multi-way join algorithm reduces the number of MR jobs. In the case of the S2 algorithm, we need $(n-1)$ MR iterations. If we use the P2 algorithm, the number of MR jobs would be reduced to $\lceil \log_2 n \rceil$. The query plan for the P2 algorithm resembles a skewed binary tree. In the case of the PM algorithm, the number of MR iterations becomes $\lceil \log_m n \rceil$. Fig 2 shows an example. We have five matrices from M_1 to M_5. Suppose that we use the three-way join algorithm. In the first MR job, $LIST\_M_{left}$ contains M_1 and M_4. $LIST\_M_{right}(M_1)$ and $LIST\_M_{right}(M_4)$ have $[M_2, M_3]$ and $[M_5]$, respectively. After the first iteration, $M_1 \bowtie_* M_2 \bowtie_* M_3$ is renamed as M_1. In the second MR job, $M_1 \bowtie_* M_4$ will be computed and the result will be generated. In this example, we have $2 = \lceil \log_3 5 \rceil$ MR jobs. If we use the S2 or the P2 algorithm, we would have $4 = (5-1)$ or $3 = \lceil \log_2 5 \rceil$ MR jobs, respectively.

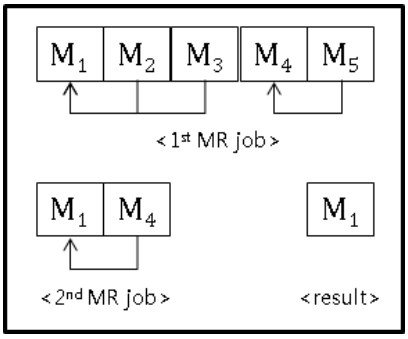


Figure 2: An example of parallel 3-way join. The number of MR iterations is $\lceil \log_3 5 \rceil$

There are several ways to implement the *map* and the *reduce* functions of the multi-way join operation. We use different *map* and *reduce* implementations for P2 and PM. In the case of P2, we can use a raw key. Suppose that we have matrices $M_1(r_1, c_1, v_1), M_2(r_2, c_2, v_2)$, and $M_3(r_3, c_3, v_3)$. Then, the first MR job matches c_1 of M_1 and r_2 of M_2 and produces the records $(r_1, c_2, v_1 * v_2)$. We can use the raw values of c_1 and r_2 without any modification; meaning that the join key is 'raw'. On the other hand, PM has to consider M_3 in a MR job. There should be two different join keys for $M_1 \bowtie_* M_2$ and $M_2 \bowtie_* M_3$. Therefore, PM use a composite key rather than the raw key. Although the composite key enables PM to reduce the number of MR jobs, it also has some weak points. One of them is illustrated in Figure 3.

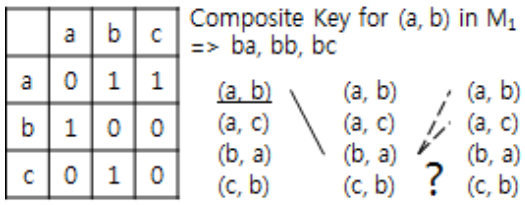
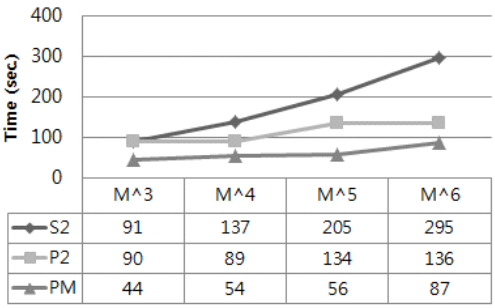


Figure 3: A mapper of PM takes a record (a, b), and generates pairs whose key is ba, bb, and bc respectively.

The use of composite key leads to record duplications. For a record (a, b) in M_1, a mapper of PM generates three keys because the mapper cannot ensure the second part of the composite key. As a result, the network overhead of PM is greater than the network overhead of P2. In the next section, we will discuss other limitations of the composite



	M^3	M^4	M^5	M^6
S2	91	137	205	295
P2	90	89	134	136
PM	44	54	56	87

Figure 4: Powers of a matrix. The intra-operation parallelism is important, and PM shows the best performance.

key.

3. EXPERIMENTS

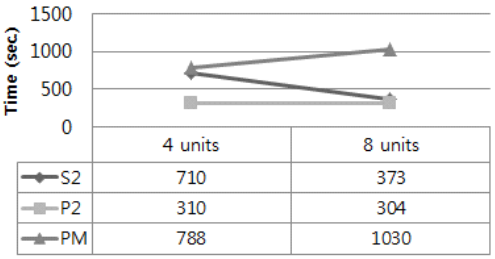
We now present the performance results for the different algorithms described in this paper. We used Amazon EC2 and Apache Whirr [2] in order to build the experimental environment. We sourced the graph data set from the Stanford Large Network Dataset Collection [3] to create an adjacency matrix.

3.1 Efficiency of the Parallel M-way Join

Our first experiment concerns the efficiency of the parallel m-way join algorithm. We created an adjacency matrix of a grpah dataset containing about 10,000 nodes, which we compute powers of the matrix. The result is described in Figure 4. The S2 and P2 algorithms similarly take 90 seconds to compute M^3 because they need the same number of MR jobs and use the same raw key. On the other hand, PM requires only 44 seconds. This result concisely shows that reducing the number of MR iterations is indeed important. As the degree of powers increases, the gap between S2 and PM gradually grows. Even P2 outperforms the S2 algorithm. This result shows the importance of the inter-operation parallelism approach.

3.2 Limitations of the Parallel M-way Join

After the experiment with the small size of matrices, we



	4 units	8 units
S2	710	373
P2	310	304
PM	788	1030

Figure 5: If the size of a matrix does not fit in a memory, PM needs disk I/O. S2 using the raw key shows continuous speed up. On the contrary, the composite key based algorithms does not achieve speed up. The performance of PM is even worse when the number of machines is increased.

increased the size of the matrix. At this point, the graph dataset contained over one million nodes. At first, we used 4 EC2 instances. Figure 5 shows the experimental results. P2 takes 310 seconds, which is the best performance among the algorithms. The response time of P2 is less than half of the response time of S2. However, PM is slower than S2. In an experiment on 8 EC2 instances, the performance of PM is even worse. Potential reasons for this are given below.

- As described in section 2.2, the parallel m-way join operation duplicates a (row, col, val) record according to the number of machines. Record duplication lead to a higher network cost between mappers and reducers.
- The composite key implementation of PM sacrifices intra-operation parallelism. In S2, we have a number of values for a join key. However, the maximum number of values for a join key in PM is the number of machines.
- We employed the Hadoop MapReduce framework [1]. Hadoop's mapper sorts local $(key, value)$ pairs according to their key. Because PM generates many records for a key, PM's sorting overhead is larger than that of S2.

Although PM reduces the number of MR iterations, the cost of the shuffle phase of PM is higher than that of S2. P2 implements the raw key. Therefore, it still shows good performance for a large dataset.

4. CONCLUSION AND FUTURE WORK

In this paper, we translated the multiplication of several matrices into a multi-way join operation among several relations. More specifically, We expanded the problem from binary multiplication to n-ary multiplication and represented a matrix as a relation consisting of (row, col, val) records and then applied the join operation of a database system. We implemented three types of algorithms: S2, P2, and PM. With several experiments on real-world graph datasets, we demonstrated that the parallel m-way join operation can improve the performance of the matrix chain multiplication process. However, using the composite key introduces a number of disadvantages, such as greater network and sorting overhead. The parallel two-way join algorithm balances the inter-operation parallelism and the intra-operation parallelism approaches because it can still use the raw key implementation. As a result, P2 shows the best performance for a large dataset.

There are many future works that can remedy the limitations when implementing the PM algorithms. If we can implement PM with the raw key, this would lead to the best performance. To do this, we plan to change the underlying record representation. This improvement should reduce the number of duplications and increase the diversity of the join key. We can also simply conduct experiments with a new MapReduce framework that does not perform sort operations in mappers.

5. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20110017480).

6. REFERENCES

[1] Apache hadoop, http://hadoop.apache.org/.

[2] Apache whirr, http://whirr.apache.org/.

[3] Stanford large network dataset collection, http://snap.stanford.edu/data/index.html.

[4] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. on Knowl. and Data Eng.*, 23:1282–1298, September 2011.

[5] R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 121–126, New York, NY, USA, 2009. ACM.

[6] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 975–986, New York, NY, USA, 2010. ACM.

[7] T. H. Cormen. *Introduction to algorithms*. MIT Press, 2001.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[9] A. Ghoting, R. Krishnamurthy, E. P. D. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *ICDE*, pages 231–242, 2011.

[10] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 229–238, Washington, DC, USA, 2009. IEEE Computer Society.

[11] J. Myung, J. Yeon, and S.-g. Lee. Sparql basic graph pattern processing with iterative mapreduce. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, MDAC '10, pages 6:1–6:6, New York, NY, USA, 2010. ACM.

[12] J. Norstad. A mapreduce algorithm for matrix multiplication, 2009.

[13] S. Seo, E. J. Yoon, J. Kim, S. Jin, J.-S. Kim, and S. Maeng. Hama: An efficient matrix computation with the mapreduce framework. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 721–726, Washington, DC, USA, 2010. IEEE Computer Society.