Distributed query optimization using reduction filters.

J.M. Morrissey, W.K. Osborn

School of Computer Science
University of Windsor
Windsor, Ontario
Canada N9B 3P4

Abstract The optimization of general queries in a distributed database management system is an important research topic. The difficulty is to select the database operations which will process the query and minimize costs. Traditional solutions include the use of heuristic strategies based on semijoin or join operations. Here, we present an approach for general queries which uses ***reduction filters***, which are based on Bloom filters, to minimize data transfers and reduce local processing costs. We discuss related work and present our algorithm — illustrating it with a simple example. The paper ends with a brief discussion of current and future work.

Related Work Bloom filters are first described in [1], and have mostly been used to improve the relational join operation [2, 3, 4, 5, 6]. Bloom filters have been applied to other relational operations [7, 8, 9] and file processing operations [10]. Mullin [11, 12] employs Bloom filters to improve the semijoin operation. His method is limited to the case where there are only two sites involved in the query and it has not been expanded to cater for general queries. Tseng and Chen [13] present a new relational operator called a hash-semijoin. They propose a method which will transform a sequence of semijoins, produced by any existing semijoin-based algorithm, into a more cost effective sequence by replacing some of the semijoins by hash-semijoins. But their approach has not been experimentally evaluated. In [14] we presented and evaluated a reduction filter algorithm, comparing its performance with a traditional semijoin-based algorithm [15] We concluded that significant improvement was possible. Our current approach is novel in that we are concerned with general queries involving many relations and attributes; we do not employ any semijoins but use reduction filters exclusively; and in addition, the evaluation methodology is quite different.

A reduction filter-based algorithm We assume a distributed relational database management system connected via a point to point network. There is no fragmentation nor replication. We consider only select-project-join (SPJ) querie, each consisting of a number of relations located at different sites which must be joined and the result made available at the query site.

0-7803-4314-X/98/$10©1998IEEE

A reduction filter is an array of bits which functions as a compact representation of the values of a join attribute in a relation. It is constructed using a hash function which set bits corresponding to the values. The filter can be used to identify tuples (in other relations) which cannot belong to the result and therefore do not need to be shipped. In essence the reduction filter achieves the same result as a semijoin but at a much lower cost

Each query is represented by a graph and an adjacency list. Each relation is usually only processed once — to minimize data transfers. However, if a filter changes during use then certain relations must be processed again — a queue is used to record this information.

The Algorithm The algorithm consists of two phases: during phase one the adjacency list is used to determine the order in which the filters are constructed and used.

Phase one:

1. Select the relation with lowest in-degree for processing.
2. Scan adjacency list to see which filters must be constructed. If a filter is already available then concurrently use it to reduce the relation and produce all required filters.
3. If a filter has changed the use the following " filter rule": if a filter for a relation changes then add that relation to the queue only if it has already been processed; it is not already on the queue; and it is not the most recently processed relation.
4. Use adjacency list to "remove edges" from query graph — that is, reduce the in-degree of each relation in the list by 1.
5. Mark relation as processed.

Repeat all steps until each relation has been processed once.

Phase two:

1. Remove relation from queue.
2. Reduce relation using all appropriate filters.
3. If a filter changes then use the "filter rule".

Repeat all steps until the queue is empty.

A simple example: We have the following relations — the arrows indicate the only tuples which can be part of the final join at the query site:

R_1	A	B	C
→	1	2	4
	2	2	5
	3	3	6

R_2	A	D	E	F
→	1	2	4	3
	2	2	5	5
	3	3	6	7
	4	4	7	9

R_3	B	D	E
→	2	2	4
	3	3	6
	4	4	9

R_4	C	G
→	4	2
	5	3
	7	4

R_5	F	H
→	3	4
→	3	7
	7	7

The query is processed as follows[1]

- R_4 is selected; a filter for attribute C is produced (4, 5, 7); the in-degree of R_1 is

[1] The adjacency list is not shown since space is not available!

reduced by 1 and R_4 is marked as processed.

- R_5 is selected; a filter for attribute F is produced (3, 7); the in-degree of R_2 is reduced by 1 and R_5 is marked as processed.
- R_1 is selected and reduced using the C filter to produce a new R_1 and filters for A, B and C — as shown.

R_1

A	B	C
1	2	4
2	2	5

A: 1, 2
B: 2
C: 4, 5

The filter for C has changed so R4 is placed on the queue. The in-degrees of R_2 and R_3 are reduced by 1 and R_1 is marked as processed.

- R_2 is selected and reduced using the A and F filters to produce a new R_2 and filters for A, D, E and F — as shown.

R_2

A	D	E	F
1	2	4	3

A: 1
D: 2
E: 4
F: 3

The filters for A and F have changed so R_1 and R_5 are placed on the queue. The in-degree of R_3 is reduced by 2 and R_2 is marked as processed.

- R_3 is selected and reduced using the B, D and E filters to produce a new R_3 and filters for B, D and E — as shown.

R_3

B	D	E
2	2	4

B: 2
D: 2
E: 4

No filters have changed, so no relation is added to the queue. All relations have been processed so phase one is complete.

- The queue contains R_4, R_1 and R_5. Each is processed in turn to produce the following reduced relations:

R_1

A	B	C
1	2	4

R_4

C	G
4	2

R_5

F	H
3	4
3	7

During processing no filter is altered, the queue is emptied and the algorithm stops.

We are currently evaluating the algorithm and initial results are promising. To evaluate, we compare our method not against another algorithm but against the effects of a "full reducer" — an algorithm which reduces the relations to just those tuples which can be part of the final result. In our example, the algorithm does fully reduce the relations. We believe that this method is more objective and gives a better assessment of the utility of the algorithm.

We are currently investigating how collisions affect the operation of the algorithm. We are also testing the use of multiple reduction filters, where each is based on a different hash function, as a method of eliminating the unfavorable consequences of collisions.

Full details of the algorithm and the results of the evaluation and all related experiments will be presented at conference time.

References

[1] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Comm. ACM*, vol. 13(7), pp. 422–426, 1970.

[2] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen, "The GAMMA database machine project," *IEEE Transactions on Knowledge and data engineering*, pp. 44–62, 1990.

[3] G. Z. Qadah, *Filter-based join algorithms on uniprocessor and distributed-memory multiprocessor database machines*, vol. 303 of *Lecture Notes in Computer Science*, pp. 388–413. Springer-Verlag, 1988.

[4] G. Z. Qadah and K. Irani, "The join algorithms on a shared-memory multiprocessor database machine," *IEEE Transactions on Software Engineering*, pp. 1168–1683, 88.

[5] P. Valduriez and G. Gardarin, "Join and semijoin algorithms for a multiprocessor database machine," *ACM Transactions on database systems*, pp. 133–161, 1984.

[6] J. Mullin, "Estimating the size of a relational join," *Information Systems*, vol. 18(3), pp. 189–196, 1993.

[7] G. Graefe and K. Ward, "Dynamic query evaluation plans," in *ACM SIGMOD*, pp. 358–366, 1989.

[8] G. Graefe, "Query evaluation techniques for large databases," *ACM computing surveys*, pp. 73–170, 1993.

[9] C. Mohan, D. Haderle, Y. Wang, and J. Cheng, *Single table access using multiple indexes: optimization, execution and concurrency control techniques.*, vol. 416 of *Lecture Notes in Computer Science*, pp. 29–43. Springer-Verlag, 1990.

[10] D. Severance and G. Lohman, "Differential files: their application to the maintenance of large databases," *ACM Transactions on database systems*, pp. 257–267, 1976.

[11] J. Mullin, "(1983) A second look at bloom filters," *Comm. ACM*, vol. 26(8), pp. 570–571, 1983.

[12] J. Mullin, "Optimal semijoins for distributed database systems," *IEEE trans. on software eng.*, vol. 16(5), pp. 558–560, 1990.

[13] J. Tseng and A. P. Chen, "Improving distributed query processing by hash-semijoins," *Journal of Information Science and Engineering*, vol. 8, pp. 525–540, 1992.

[14] J. Morrissey and W. Osborn, "Experiments with the use of reduction filters in distributed query optimization," in *Proceedings of the 9th IASTED International Conference on Parallel and Distributed Systems*, (Georgetown University, Washington), pp. 327–330, 1997.

[15] J. Morrissey and W. Bealor, "Minimizing data transfers in distributed query processing: a comparative study and evaluation," *The Computer Journal*, vol. 39(8), 1997.